C/C++ התוכנית הראשונה שלי

אלן ר. נייבאואר



מכיל תקליטון קומפיילר C - חינם
חיסכון של מאות דולרים

- מתאים במיוחד למתחילים
- הדרכה מלאה ודוגמאות בשיטת צעד-אחר-צעד
- כולל שאלות למבחן עצמי - המסייעות לך לזכור את החומר הנלמד

MULTI SYSTEM BUG

SYBEX

C/C++

התוכנית הראשונה שלי

אלאן ר. נייבאור





ספר זה תורגם מ: Your First C/C ++ Program

ISBN: 0-7821-1414-8

Authorized Translation from English Language Edition

תרגום: רונן אליאב

הפקה: לילי צראף - באג מולטיסיסטם בע"מ

עריכה לשונית: שולה גרינברגר - באג מולטיסיסטם בע"מ

סדר ועיצוב: מיקסם טכנולוגיות בע"מ

דפוס: טופרינט בע"מ

כריכייה: כריכיית אהרון

הודפס בישראל כסלו תשנ"ד—נובמבר 1994

שיווק והפצה:

באג מולטיסיסטם בע"מ

רחוב כנרת 13 בני ברק טל: 03-5794711, פקס: 03- 5708174

תוכן העניינים

הקדמה

שכח את כל אותם סיפורים ששמעת על הקשיים הרבים הכרוכים בתכנות מחשבים. התעלם מכל אותן מעשיות על התמודדות ומאמץ אדירים. אינך צריך להיות מוטרד כלל אם אינך בעל תואר דוקטור למתמטיקה או למדעים.

לימוד תכנות מחשבים, וכתיבת התוכניות בפועל, יכולים להיות מהנים. אם אתה בעל מחשבה לוגית, אם אתה אוהב לפתור חידות ובעיות, או אם ברצונך לשלוט במחשב שברשותך במקום שהמחשב ישלוט בך, הרי שאתה מועמד מושלם ללימוד תכנות.

אתה אוחז בידך את הספר שיכול להכניס אותך לעניינים בצורה הטובה ביותר. ספר זה נועד למתחילים, לעושים את צעדיהם הראשונים בתכנות. למעשה, הנחת המוצא של הספר היא שאינך יודע דבר וחצי דבר אודות תכנות מחשבים. (הספר מתאים במיוחד גם למתכנתים מנוסים העוברים לשפת C ולשפת ++C משפות תכנות אחרות, כגון BASIC, פסקל, או אפילו שפות מקרו המשמשות בתוכניות כמו WordPerfect, Lotus ו-Excel).

לאמיתו של דבר, כל שנדרש כדי ללמוד תכנות הוא הספר הזה ורצון ללמוד. התקליטון המצורף לספר מכיל אפילו מהדר (קומפיילר) ועורך תוכניות, בעזרתם תוכל לכתוב תוכניות משלך.

תוכנית ++C/C הראשונה שלך מתמקד בגרסאות של שפת התכנות C הידועות בכינויים תקן K&R ותקן ANSI C, וכן במאפיינים המרכזיים של גרסת ++C. ניתן ליישם את כל הטכניקות שתלמד בספר זה בתוכניות C ובתוכניות ++C. אם אתה מתעניין בשפת ++C, חפש את הקטעים, ההערות והערות השוליים המתייחסות במיוחד לגרסת העילית הזו של שפת התכנות C.

מה מעניק לך הספר

בעזרת ספר זה תלמד, צעד אחר צעד, כיצד לכתוב תוכניות. תמצא בספר הסברים בהירים, קלים להבנה, וכן שפע של תרשימים ודוגמאות. על ידי קריאת ההסברים ועיון בתרשימים ובתוכניות ההדגמה, תלמד להכיר את כל הנושאים הקשורים בתכנות, על-פי הסדר המתאים. כל אחד מפרקי הספר מסתיים בשאלות חזרה שימושיות ובתרגילים מעשיים.

הספר נועד, אמנם, למתכנת המתחיל, אך הוא משמש גם כספר מבוא מקיף. לא החסרנו דבר. בעזרת ספר זה תלמד לא רק שפת תכנות, אלא גם תגלה את ההיגיון שבעיצוב תוכניות ובפתרון בעיות.

בפרקים 1 ו-2 תלמד על תהליך התכנות, וכן על המבנה והתכונות של תוכניות C ותוכניות ++C. בפרק 3 תלמד כיצד לתקשר עם תוכנית, תוך שימוש במשתנים ובקבועים.

בהמשך, תלמד בפרק 4 כיצד להציג מידע על-גבי הצג, ובפרק 5 תגלה כיצד להזין מידע מהמקלדת. בפרק 6 תלמד איך לכתוב תוכניות, תוך שימוש בסימני פעולה (אופרטורים) לביצוע חישובים.

בפרק 7, העוסק בפונקציות, תגלה כיצד לבנות את תוכניותיך בקטעים, תוך כדי חלוקת התוכנית למקטעים שקל לטפל בהם. פרק 8 עוסק בנושא קבלת החלטות, ופרק 9 מדגים כיצד לחזור על הוראות באמצעות לולאות.

לאחר שרכשת כישורי תכנות, תלמד בפרק 10 על מערכים ומחרוזות, ובפרק 11 על מצביעים ומבנים. בפרק 12 תלמד כיצד לקרוא ולכתוב קבצים, וכיצד לשגר נתונים למדפסת.

כדי לשלב זה בזה את הכישורים שרכשת, מציג פרק 13 תוכנית הדגמה מקיפה המשמשת לבניית יישום של מסד נתונים. מקריאת הפרק ומעיון בקוד התוכנית תלמד כיצד יש לבנות יישומים מקצועיים.

הנספח בסוף הספר מסביר כיצד להתקין את הקומפיילר ואת עורך התוכניות שבתקליטון המצורף, וכיצד להשתמש בהם, וכן כיצד לגשת לתוכניות ההדגמה ולפתרונות לתרגילים, הנמצאים אף הם על-גבי התקליטון.

לאחר שתסיים לקרוא את פרקים 1 ו-2, בצע את ההוראות הכלולות בנספח לצורך התקנת התוכניות שעל התקליטון. לאחר ההתקנה תוכל לעקוב, תוך כדי קריאת הפרקים הבאים, אחר ההדגמות שבתוכניות. אנו בטוחים שתוכל במהרה לכתוב תוכניות משלך, לשימוש במשרד, בביה"ס או בבית.

כלי תכנות מושלם

אם יש כבר ברשותך קומפיילר C או ++C, תוכל להיעזר בו ללימוד שפת התכנות C ולכתוב את התוכניות המוצגות בספר. אך גם אם אין ברשותך קומפיילר משלך, כל מה שאתה זקוק לו כלול בתקליטון המצורף לספר.

התקליטון המצורף לספר זה כולל יותר מאשר תוכניות הדגמה בלבד. הוא מכיל קומפיילר C מלא, קשר, ספריית פונקציות מקיפה, וכן חוברת עזר בת 80 עמודים, אותה תוכל להדפיס בעזרת מעבד תמלילים או באמצעות פקודת Print של Dos.

התקליטון כולל גם עורך מלל לכתיבת תוכניות ולעריכתן, ואפילו מאגד תוכניות, שיסייע לך אם תרצה ללמוד בשלב מאוחר יותר את שפת התכנות Assembly. בתקליטון תמצא גם עותקים של כל תוכניות ההדגמה המופיעות בספר. תוכל לקמפל ולהריץ את התוכניות בצורתן הנוכחית, או לפתוח אותן בעזרת העורך, כדי להתעמק בהן או להשתמש בהן כבסיס לבניית תוכניות משלך.

1

במה דברים אמורים?

אנשים בוחרים ללמוד תכנות מחשבים מסיבות שונות. בראש ובראשונה, כמובן, מדובר בכסף. מרבית תוכניות המחשב נכתבות כאמצעי להרוויח כסף. תוכניות מסוימות נכתבות עבור קהל יעד או ענף מסוימים, בעוד שתוכניות אחרות נכתבות עבור הקהל הרחב של משתמשי המחשב. כך או כך, המטרה דומה - ליצור הצלחה מסחרית. ייתכן שלא תכתוב לעולם תוכנית מחשב שתהפוך "להיט" עולמי, אך תוכל להרוויח כסף מכתיבת תוכניות מחשב עבור משתמשים אחרים.

ייתכן שברצונך לכתוב תוכניות מחשב בגלל האתגר והחוויה הכרוכים במשימה, בדיוק מאותה סיבה שאנשים אחרים מטפסים על הרים, פותרים תשבצים או מנסים להתקבל לנבחרת האולימפית. אם אתה מתכנת מטיפוס כזה, אתה ודאי שואב סיפוק רב מעצם כתיבת התוכנית, מתחילתה ועד סופה. תוכנית מחשב היא מעין חידה, תרגיל שכלי, בו אתה מתמודד נגד המחשב להשגת שליטה מלאה.

ייתכן שאתה עוסק בתכנות כדי לפתור בעיה מיוחדת. עליך לבצע מטלה כלשהי בעזרת המחשב, ואינך מוצא תוכנית מסחרית העונה לצרכיך, או שאינך יכול להרשות לעצמך לרכוש תוכנית כזו. זו יכולה להיות תוכנית פשוטה ביותר, כמו תוכנית לאיסוף ותיעוד של מרשמים לבישול, או תוכנית מורכבת לפתרון בעיה עסקית או הנדסית ייחודית.

תהא אשר תהא הסיבה לכך שאתה מעוניין לתכנת את המחשב, העיסוק בתכנות מעניק לך תחושה של הישג אישי והצלחה. קל להתמכר לתחושת הסיפוק שחשים כאשר התוכנית פועלת כהלכה. מאידך, התסכול שחשים כאשר התוכנית אינה פועלת, עלול להיות הרסני. אך כמו כל אתגר - זה שווה את המאמץ.

תוכנית מחשב

קרוב לוודאי שכבר התנסית בהרצת תוכניות מחשב - מעבד תמלילים, תוכנית גיליון נתונים אלקטרוני, מסד נתונים, או אפילו משחק מחשב אחד או שניים. אך כאשר אתה מריץ תוכנית מחשב, אתה מתבונן בה מבחוץ בלבד. אינך רואה מה עושה התוכנית באמת, אלא רק את תוצאות פעולתה.

כאשר אתה כותב תוכנית מחשב, אתה מתבונן בה מבפנים. אתה יודע כיצד התוכנית פועלת, ומדוע היא פועלת כך ולא אחרת. זוהי דרך שונה לגמרי ומלהיבה יותר, להתבונן בתוכנית מחשב. אין זה משנה עד כמה אתה בעל ניסיון בהרצת תוכניות, או עד כמה אתה מתמצא במחשבים, כתיבת תוכנית מחשב מציבה בפניך אתגרים חדשים לחלוטין.

האם הזדמן לך פעם להדריך מישהו המחפש כתובת כלשהי? האם לימדת פעם מישהו כיצד לבצע דבר-מה? זה מה שעושה תוכנית מחשב. תוכנית מחשב אינה אלא סדרה של הוראות למחשב, המורות לו *בדיוק מה עליו לעשות*, ו*באיזה סדר פעולות*, *בשפה שהמחשב מבין*. *זהו זה.*

כאשר הוראה כלשהי אינה מובנת למחשב, הוא מודיע על כך באמצעות הצגת *הודעת שגיאה*. כל שעליך לעשות הוא לנסות שוב, לשכתב את ההוראה כך שהמחשב יוכל לעקוב אחריה. לעתים המחשב יכול לעקוב אחר הוראותיך אפילו כשההוראות שגויות! האם קרה לך שביקשת הוראות כיצד להגיע למקום מסוים, ולמרות שביצעת אותן במדויק לא הגעת למקום הרצוי? דבר דומה יכול לקרות גם בתוכנית מחשב. אלו הן הבעיות הקשות ביותר לפתרון, מכיוון שאתה עשוי שלא לדעת כלל על קיומן.

עתה, הבה נמשיך ונבחן את ההגדרה של תוכנית מחשב.

ההוראות מורות למחשב מה עליו לעשות בדיוק. מלת המפתח היא *בדיוק*. כל הוראה חייבת להיות מדויקת. לא ניתן לדלג על שלבים כלשהם. לדוגמה, נניח שמישהו שואל אותך כיצד להגיע לסוכנות הדואר השכונתית. אתה עונה: "המשך ישר לאורך שני גושי בניינים ופנה שמאלה ברחוב פרנקלין". אתה מניח, כדבר מובן מאליו, שאותו אדם יראה את סוכנות הדואר וייכנס אליה.

הערות ++C

++C היא שפה נוהלית פחות משפות תכנות אחרות. הוראות מסוימות מתייחסות ל"עצמים" ול"אירועים", במקום להציג רצף של צעדים. עם זאת, ההוראות, בין אם הן נוהליות ובין אם לאו, חייבות להיות מפורשות ומלאות.

בעבודה עם המחשב, לעומת זאת, אינך יכול להניח שדבר-מה הוא מובן מאליו. לכן, אם עליך להדריך את המחשב כיצד למצוא את סוכנות הדואר, ההוראות שתמסור יהיו כדלקמן: המשך ישר לאורך שני גושי בניינים. בצע פנייה של 90 מעלות לכיוון שמאל. המשך להתקדם עוד 20 מטרים. בצע פנייה של 90 מעלות לכיוון שמאל. טפס ארבע מדרגות. פתח את הדלת והיכנס פנימה.

כאשר אתה שומר מסמך בתוכנית עיבוד תמלילים, לדוגמה, על התוכנית לדעת בדיוק אילו צעדים עליה לבצע. עליה לדעת היכן להציב את המסמך בתקליטון. עליה גם לקבל הוראות להציג הודעת אזהרה כלשהי, במקרה ששמירת המסמך החדש עלולה לשכתב מסמך שכבר שמור על-גבי התקליטון.

ההוראות חייבות להינתן בסדר הנכון. אם תורה למחשב לבצע פעולות A, B ו-C, המחשב יבצע אותן בסדר הזה. המחשב אינו יכול לחשוב "רגע, זה לא נשמע לי נכון, אולי עלי לפנות קודם לרחוב פרנקלין". עם זאת, ניתן לגרום למחשב לקבל החלטות. התוכנית יכולה להחליט "אם משכורתה גבוהה מ-$50,000, אז יש לנכות 25% מסים, ואם לא - לנכות רק 15%". אך עליך לומר למחשב מתי לקבל את ההחלטה, ומה לעשות כאשר מתקיים כל אחד מהתנאים.

לדוגמה, נבחן שוב את המקרה של שמירת מסמך חדש, ששמו זהה לשמו של מסמך שכבר קיים על-גבי התקליטון. אם התוכנית היתה שומרת את המסמך החדש, תוך כדי מחיקת המסמך הקודם, ואז שואלת אותך אם זה מה שאתה אכן רוצה לעשות - זה לא היה פתרון מוצלח. על התוכנית להציג תחילה את ההודעה, ואז להחליט מה לעשות על סמך הקלט שלך. אם תאשר את הפעולה, התוכנית תשכתב את המסמך הקיים. אם לא תאשר את הפעולה, התוכנית תבקש ממך להקליד שם חדש עבור המסמך שברצונך לשמור.

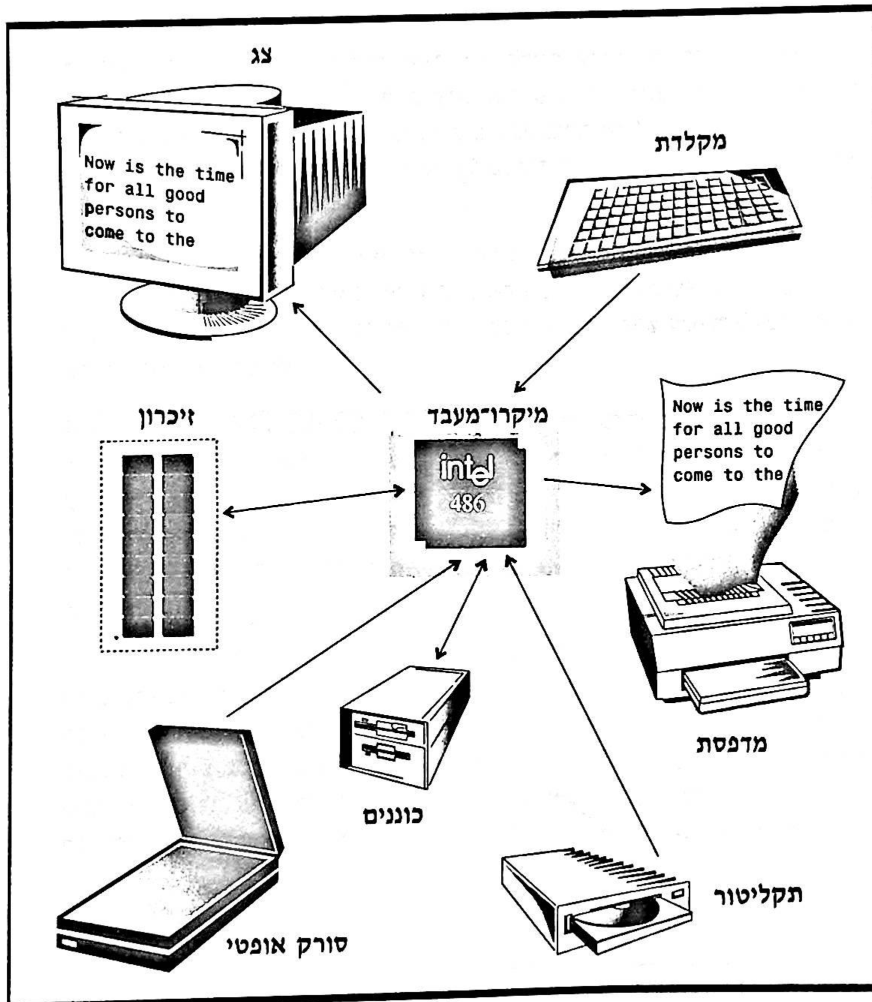
שפות תכנות

הדרישה השלישית מתוכנית מחשב היא שעליה להיות כתובה בשפה שהמחשב יכול להבין.

עמוק בתוך המחשב נמצא *מיקרו-מעבד*, מעגל משולב יחיד השולט בכל מה שמתרחש במחשב. (ראה תרשים 1.1).

כאשר התוכנית מורה למחשב להציג הודעה על-גבי הצג או להדפיסה במדפסת, המיקרו-מעבד משגר את האותות האלקטרוניים המתאימים. אותות אלה מורים למחשב היכן נמצאת ההודעה בזיכרון, ולאן עליו לשגר אותה. מרבית פעולותיו של המיקרו-מעבד אינן נראות לעין - אתה מזין את התוכנית שלך למחשב ומניח שהמיקרו-מעבד יודע את תפקידו. אולם כדי להבין כיצד פועל התכנות, עליך לדעת מעט יותר - לא הרבה - על אופן פעולתו של המיקרו-מעבד.

תרשים 1.1
המיקרו-מעבד שולט בכל מה שמתרחש במחשב.

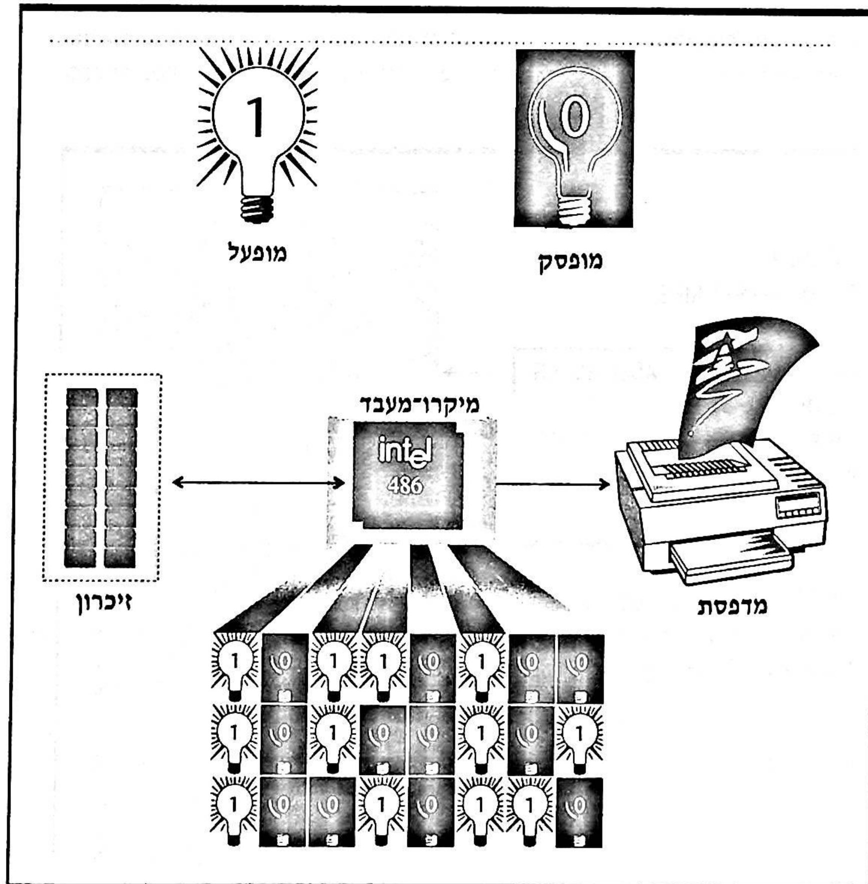


מבחינה טכנית, המיקרו-מעבד מסוגל לבצע ארבע פעולות בלבד. ביכולתו *להזיז* נתונים מכתובת זיכרון אחת לאחרת, *לשנות* נתונים בתוך כתובת זיכרון, *לקבוע* אם כתובת זיכרון כלשהי מכילה נתונים מסוימים, ו*לשנות* את רצף ההוראות שהוא מבצע. כל הפעולות האלה מבוצעות על ידי שיגור אותות אלקטרוניים, קליטתם או בקרה עליהם.

המחשב מטפל באותות האלקטרוניים באחד משני מצבים (רמות מתח); אות אלקטרוני יכול להיות מופעל או מופסק. כדי לבצע מטלה, אנו מזינים סדרה של אותות - במצב מופעל או מופסק - למיקרו-מעבד. השילוב של מצבי מופעל/מופסק המודגם בתרשים 1.2, למשל, מדפיס את התו A.

כדי לכתוב הוראות למחשב ברמה בסיסית זו, אנו משתמשים בספרה 0 כדי לייצג את המצב "מופסק", ובספרה 1 כדי לייצג את המצב "מופעל". ספרות אלה מכונות "ספרות בינאריות" (ביטים), או הוראות בינאריות, בהתבסס על שיטת המספור הבינארי (בסיס שתיים), העושה שימוש בצירופים של 0 ו-1 בלבד כדי לייצג את כל המספרים.

תרשים 1.2
סדרה של אותות אלקטרוניים במצב מופעל או מופסק מורה למיקרו-מעבד איזו מטלה לבצע - במקרה זה, לאחזר תו מהזיכרון ולשגרו למדפסת.

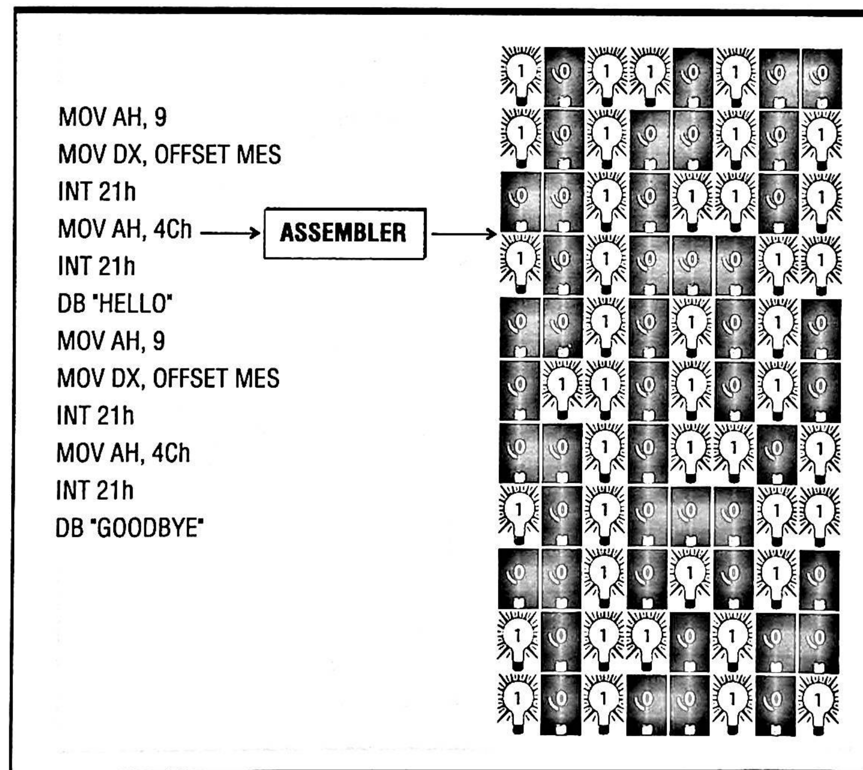


בראשית ימי המחשבים ותוכניות המחשב, תוכנית היתה מבוצעת על ידי תפעול ישיר של אותות מופעל/מופסק אלה. המיקרו-מעבד לא היה קיים עדיין. הטכנאי, ששימש כבקר, היה מציב שורה של מתגים במצב מופעל או מופסק. אך מכיוון שלביצוע יישום שלם נדרשים אלפי אותות בודדים, כתיבת תוכנית מחשב היתה מטלה שגוזלת זמן רב.

עם התפתחות המחשבים, נפתחה הדרך להזין תוכנית למחשב כיחידה אחת, ואז לגרום למחשב לבצע את ההוראות הכלולות בה. עדיין היה צורך באלפי ספרות 0 ו-1 בודדות, עד שפותחה שפת התכנות Assembly.

שפת Assembly עושה שימוש *בקוד מנמוני (mnemonic)* כדי לייצג מטלות מחשב המתייחסות ישירות לחומרה. קוד מנמוני הוא מלה או קיצור שקל לזכרם, המייצגים מטלת מיקרו-מעבד מלאה. לדוגמה, הקוד MOV מורה למחשב להזיז (move) מידע כלשהו ממיקום מסוים בזיכרון למיקום אחר. הקוד JMP מורה למחשב לדלג (jump) למיקום אחר בזיכרון. וכך, במקום לכתוב סדרה של אפסים ואחדים, המתכנת בשפת Assembly יכול להשתמש בקודים כגון MOV ו-JMP, כשכל קוד מייצג שמונה הוראות בינאריות, או יותר.

כפי שניתן לראות בתרשים 1.3, תוכנית הנקראת Assembler מתרגמת את הקודים האלה לאותות אלקטרוניים אותם יכול המחשב להבין. מכיוון שכל קוד מתייחס ישירות לפעולות



תרשים 1.3
התוכנית Assembler ממירה את ההוראות בשפת Assembly לקוד בינארי.

הפנימיות של המיקרו-מעבד, התוכניות הנכתבות בשפה זו מורצות במהירות רבה מאוד. אולם השימוש בשפת Assembly עדיין גוזל זמן ודורש מספר גדול של קודים.

מרבית המתכנתים משתמשים היום *בשפת תכנות עילית*, שהוראותיה מורכבות ממלים המובנות לבני-אדם, ואינן רק קודים מנמוניים או סדרות של 0 ו-1. כל מלה מייצגת פעולה מעשית שלמה, ולא רק מטלת מיקרו-מעבד אחת. לדוגמה, הפונקציה puts בשפת התכנות C מורה למחשב להציג מידע מסוים על המסך. כדי לבצע אותה פעולה בשפת Assembly היו נדרשים מספר קודים מנמוניים, ואולי מאות אותות בינאריים אלקטרוניים.

תרשים 1.4 מדגים הוראה פשוטה בשפת C/C++, להצגת מלה על-גבי המסך, ואת ההוראות המקבילות לה בשפת Assembly ובקוד בינארי. איזו משפות התכנות נראית לך קלה יותר לקריאה ולכתיבה?

תרשים 1.4
הוראות מקבילות בשפת C, בשפת Assembly ובקוד בינארי.

בשפת C

```
puts("SYBEX")
```

בשפת Assembly

```
     MOV  AH, 9
     MOV  DX, OFFSET NAME
     INT  21h
     MOV  AG, 4Ch
     INT  21h
NAME DB   "SYBEX"
```

בקוד בינארי

```
101101000000100110111010000000100001011
1100110100100001101101000100110011001101
0010000101010011010110010100001001000101
01011000
```

כמובן שהמחשב אינו מבין באמת מה פירוש הפקודה PUTS או כל פקודה אחרת בשפת תכנות עילית. קודם שהמחשב יוכל לבצע את הפקודה, יש לתרגם את הפקודה לשפת המחשב עצמו - קוד בינארי.

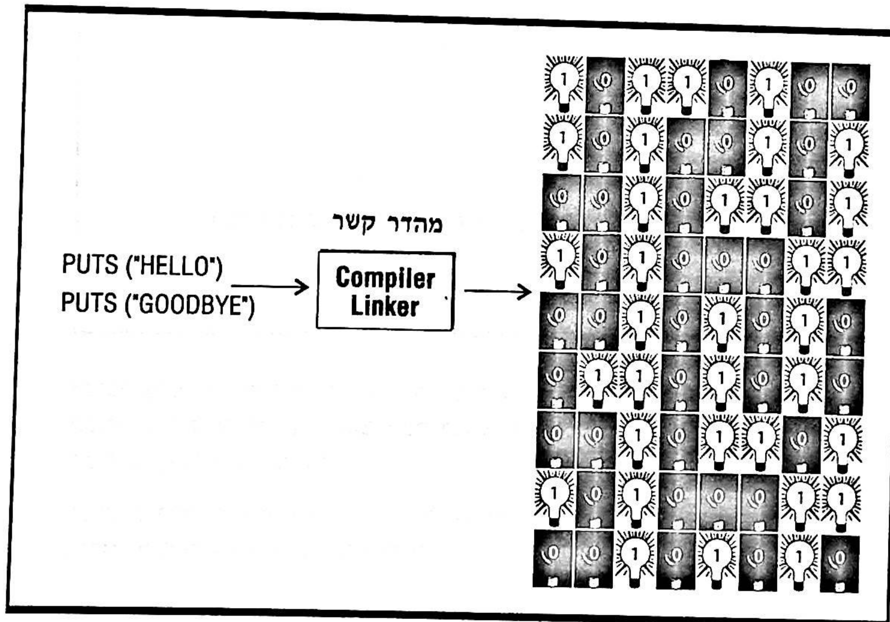
התרגום ממלים אנושיות להוראות בינאריות יכול להתבצע באחת משתי דרכים. על ידי *הידור (קומפילציה)*, או על ידי *תרגמן*.

מהדרים (קומפיילרים)

פעולת הקימפול מתרגמת את התוכנית כולה כיחידה אחת ושומרת אותה על-גבי דיסק, כדי שניתן יהיה להריץ את התוכנית במועד מאוחר יותר. כדי לדמיין את אופן הפעולה של התהליך, הבה נתבונן במצב מציאותי.

נניח שאתה מכין דו"ח בשפה האנגלית, אותו עליך להציג לפני הפרלמנט הצרפתי. אתה שוכר מתרגם שיתרגם את הדו"ח לצרפתית, מכין עותקים במספר הדרוש, ואז מפיץ את העותקים. במקרה שהמתרגם מגלה שגיאות דקדוק בדו"ח הוא מפסיק את עבודתו ומדווח לך על כל שגיאה, ועליך לתקן את השגיאה קודם שהמתרגם יוכל להמשיך בעבודת התרגום. עם זאת, אם תרצה להשתמש באותו הדו"ח בשנה שלאחר מכן, כל שיהיה לעשות הוא להעתיק אותו ולהפיץ את התרגום. לא תזדקק עוד לשירותו של המתרגם.

כפי שמודגם בתרשים 1.5, זוהי הפעולה שמבצע הקומפיילר בשפת המחשב. הקומפיילר הוא תוכנית מחשב, שבעזרת תוכנית נוספת הנקראת קשר - linker, הופכת את כל ההוראות שנתת לקוד בינארי, כדי שניתן יהיה להריץ את התוכנית. הקומפיילר מוודא שהתוכנית עומדת בכללים שמכתיבה שפת C או ++C, ואז יוצר *קובץ עצמים*, מעין צורת ביניים של התוכנית. תוך כדי התהליך, הקומפיילר יודיע לך אם נתקל בהוראה שאינו מבין. עליך לתקן את הבעיה ולנסות פעם נוספת. הקשר ממיר אז את קוד העצמים לתוכנית שניתן להריצה. תהליך זה אינו מריץ את התוכנית, אלא מתרגם אותה בלבד.



תרשים 1.5
הקומפיילר והקשר ממירים את ההוראות משפה עילית לקוד בינארי.

הערות ++C

בהמשך תלמד ששפת ++C היא מעין גרסה משופרת של שפת C. פירוש הדבר שאם קומפיילר C יכול לקמפל את התוכנית שלך, גם קומפיילר ++C יוכל לקמפל אותה.

כאשר אתה משתמש בקומפיילר, התוכנית קיימת בשלושה מצבים. הפקודות בשפת C מאוחסנות בקובץ מלל הנקרא *קוד מקור*. ניתן להדפיס את הקובץ הזה ולקרוא אותו, בדומה לכל מסמך של מעבד תמלילים. באפשרותך לערוך את הקובץ כדי להכניס שינויים בתוכנית. התוכנית המקומפלת מאוחסנת *בקובץ עצמים*, ואילו התוצאות הסופיות מאוחסנות *בקובץ הפעלה*, שאותו מריצים.

C, ++C, פסקל, קובול ו-FORTRAN הן דוגמאות לשפות מחשב מקומפלות.

סיומות של שמות קבצים

במרבית הקומפיילרים, הסיומת של שם הקובץ המכיל את קוד המקור צריכה להיות C., ואילו קובץ העצמים מקבל את הסיומת OBJ.. הקומפיילר המצורף לספר זה דורש שסיומת קוד המקור תהיה C., ושהסיומת של קובצי העצמים תהיה O.. הקומפיילר שברשותך עשוי לדרוש סיומות אחרות.

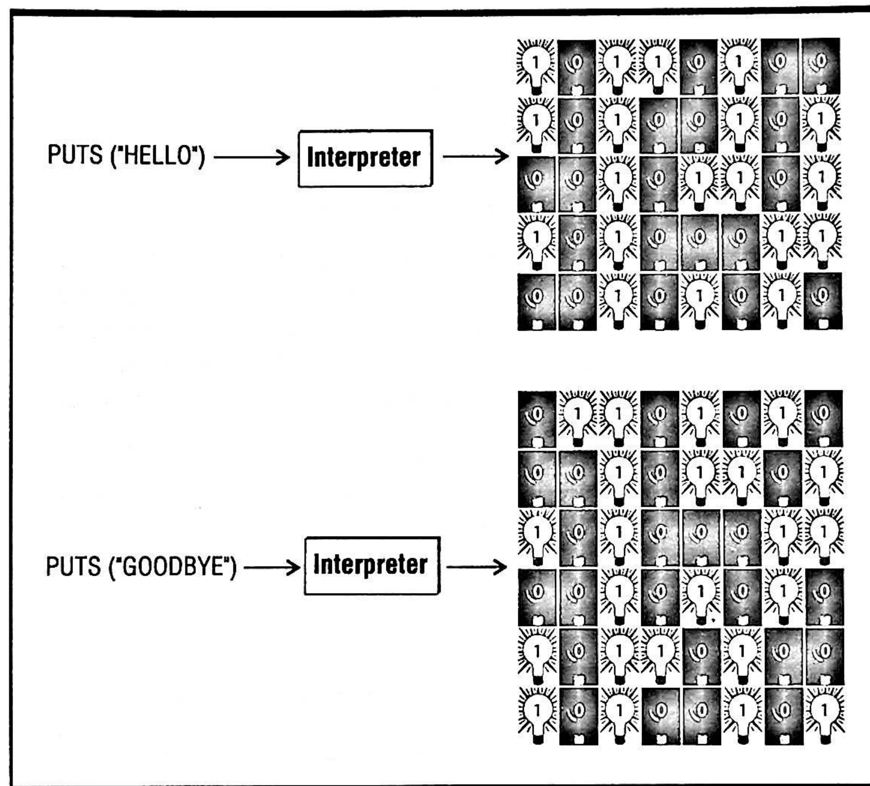
תרגמנים (interpreters)

התרגמן מתרגם כל הוראה למחשב תוך כדי הרצתה. הבה נבחן שוב את הדוגמה של הדו"ח לפרלמנט הצרפתי.

אתה כותב את הדו"ח באנגלית ושוכר מתורגמן. המתורגמן ממיר את המשפט הראשון לצרפתית ומקריא אותו באוזני חברי הפרלמנט הצרפתי המאזינים. לאחר מכן המתורגמן מתרגם ומקריא את המשפט השני, וכך הלאה, עד סיום הדו"ח. זכור שהדו"ח קיים בפועל רק בשפה האנגלית. אם בעוד שנה תרצה שדובר צרפתית יקרא את הדו"ח, תצטרך לשכור שוב מתורגמן ולחזור על כל התהליך.

תרגמן של מחשב פועל בדרך דומה. כפי שמודגם בתרשים 1.6, הוא מתרגם ומריץ את ההוראות, זו אחר זו. בעבודה עם תרגמן, התוכנית קיימת רק בגרסתה המקורית, בשפת המקור. שפת התכנות BASIC, המסופקת חינם עם MS-DOS, היא דוגמה לשפה העושה שימוש בתרגמן.

תרשים 1.6
התרגמן מתרגם הוראות משפת תכנות עילית לקוד בינארי בכל פעם שהתוכנית מורצת.



מדוע שמישהו ירצה להשתמש בתרגמן? כאשר אתה לומד תכנות, התרגמן מאפשר לך לכתוב ולבדוק את התוכנית שלך, שורה אחר שורה. שפות מקומפלות, לעומת זאת, מחייבות אותך להשלים את התוכנית כולה, או לפחות קטעים עצמאיים, בני-הרצה, קודם שניתן יהיה להריץ ולבדוק אותם.

אם כך, מדוע להשתמש בקומפיילר? בגלל העובדה שתרגמנים הופכים את המכניקה של כתיבת תוכנית לפשוטה וקלה, משתמשים רבים מזניחים את התכנון והעיצוב הדרושים כדי ליצור תוכנית שפועלת כהלכה. הם מזנקים ישר למים העמוקים, מנסים לכתוב תוכנית "ביעף", ומבזבזים שעות ארוכות בתיקון שגיאות בשיטת ניסוי וטעייה. אם אתה לומד תכנות לראשונה, כדאי לרכוש הרגלי עבודה נכונים כבר מההתחלה.

תוכניות הכתובות בשפות העושות שימוש בתרגמנים רצות לאט יותר. עליך להטעין את התרגמן לזיכרון, ואז לתרגם ולהריץ כל שורה בתוכנית בנפרד. קומפיילרים, לעומת זאת, ממירים את התוכנית בבת אחת. כאשר אתה מריץ תוכנית מקומפלת, היא כבר קיימת בצורת הוראות בינאריות שהמחשב יכול להריצן ישירות.

יתרונותיה של C/C++

שפת התכנות C היא שפה מקומפלת. זהו אוסף של פקודות ופונקציות, בצורת מלים מוכרות, המומר לקוד בינארי אותו יכול המחשב להריץ. בשנים האחרונות הפכה C לשפת התכנות המועדפת על ידי מרבית המתכנתים, וזאת בגלל שלוש סיבות עיקריות - מהירות, התאמה למחשבים שונים ובקרה.

הערות C++

זכור שכל המידע הכלול בספר זה מתייחס הן לשפת C והן לשפת C++. במקום לציין שוב ושוב את הביטוי C/C++, נתייחס בדרך-כלל לשפת C בלבד. אין פירוש הדבר שאתה לומד את שפת C בלבד - אלא שאתה לומד את שתי השפות בעת ובעונה אחת.

מהירות

C נחשבת "קרובה" יותר לשפת Assembly מכל שפת תכנות עילית אחרת, מכיוון שפקודות C מסוימות מתייחסות ישירות לחומרת המחשב. עובדה זו מאפשרת הרצה מהירה של תוכניות C מקומפלות. למעשה, מהירות ההרצה היא כה גבוהה, עד שניתן להשתמש בשפת C לכתיבת מערכות הפעלה, יישומי תקשורת ויישומים הנדסיים, ואפילו קומפיילרים.

בנוסף, מרבית הקומפיילרים של C יוצרים קוד ברמת *אופטימיזציה גבוהה*. זוכר את ההוראות הבינאריות להן זקוק המחשב? ככל שמספר ההוראות שהקומפיילר חייב ליצור נמוך יותר - רמת האופטימיזציה של הקוד גבוהה יותר - והתוכנית מהירה יותר. קומפיילרים של שפות מסוימות יוצרים קוד ברמת אופטימיזציה נמוכה, ולכן התוכניות שנכתבות בשפות אלה איטיות יותר.

התאמה למחשבים שונים

ניתן ליצור תוכניות מהירות גם בשפת Assembly. אולם, המנמוניקה של שפת Assembly שונה עבור כל משפחה של מיקרו-מעבדים. אם כתבת תוכנית בשפת Assembly עבור מחשב אישי של IBM או תואם IBM, וברצונך להשתמש באותה תוכנית על מחשב מקינטוש של Apple, יהיה עליך לשכתב את התוכנית כולה.

שפת C עושה שימוש בסדרת פקודות תקנית ואחידה למדי. באופן כללי, עליך לכתוב את התוכנית פעם אחת בלבד, ללא קשר *לפלטפורמה* (מחשב או מערכת הפעלה) עליה אתה מתכוון להריץ את התוכנית. בעוד שקוד המקור אינו משתנה, תזדקק לשני קומפיילרים - האחד לצורך תרגום התוכנית להוראות בינאריות אותן יכול מחשב IBM להבין, וקומפיילר

נוסף לצורך תרגום התוכנית לסוג ההוראות הבינאריות של מחשבי Apple. אבל את התוכנית עצמה עליך לכתוב פעם אחת בלבד.

פירוש הדבר הוא שאם למדת לתכנת בשפת C, שוב אינך צריך ללמוד שפת תכנות נוספת כדי לתכנת מחשב אחר. באפשרותך להעביר את כישוריך מפלטפורמה אחת לאחרת, ללא צורך בהכשרה מחדש. אחרי הכל, אינך יודע לאן עשויים כישורי התכנות שתרכוש להוביל אותך, וכדאי להיות מוכן.

בקרה

C היא אמנם שפה קלה ללימוד, אך גם לה יש דרישות משלה. בשפת BASIC, העושה שימוש בתרגמן, אתה יכול להתיישב ליד המחשב ולכתוב תוכנית ללא הכנה מוקדמת. בשפת C הדברים אינם כה פשוטים. לשפת C יש מבנה, דרך מסוימת בה נעשים הדברים, שמחייבת אותך לחשיבה לוגית. ניתן אמנם לדלג על דרישות מבניות רבות ולכתוב תוכנית "מהירה וגסה" שאינה עושה הרבה. אך כדי לכתוב יישום רציני בשפת C, יש להכין תחילה שיעורי בית.

עם זאת, המבנה הזה כלל אינו מהווה סרבול, אלא מקל על העיצוב של תוכניות C, כמו גם על תחזוקתן ועל איתור ותיקון השגיאות.

ספריות פונקציות

שפת C עצמה אינה כוללת פקודות רבות. בניגוד לשפות תכנות אחרות, C אינה כוללת פקודות מובנות לקלט ופלט של נתונים, או לטיפול במחרוזות. (מחרוזת היא סדרה של תווים היוצרים מלה או משפט). ההגדרה המקורית של שפת C למשל כללה 27 מלות מפתח - פקודות, בלבד.

כוחה של שפת C נובע מספריות הפונקציות המסופקות עם הקומפיילר. *פונקציה* היא סדרה של הוראות המבצעת מטלה מסוימת. *ספרייה* היא קובץ דיסק נפרד, המסופק עם הקומפיילר, ומכיל פונקציות לביצוע מטלות שכיחות, כדי שלא יהיה עליך לכתוב אותן בעצמך.

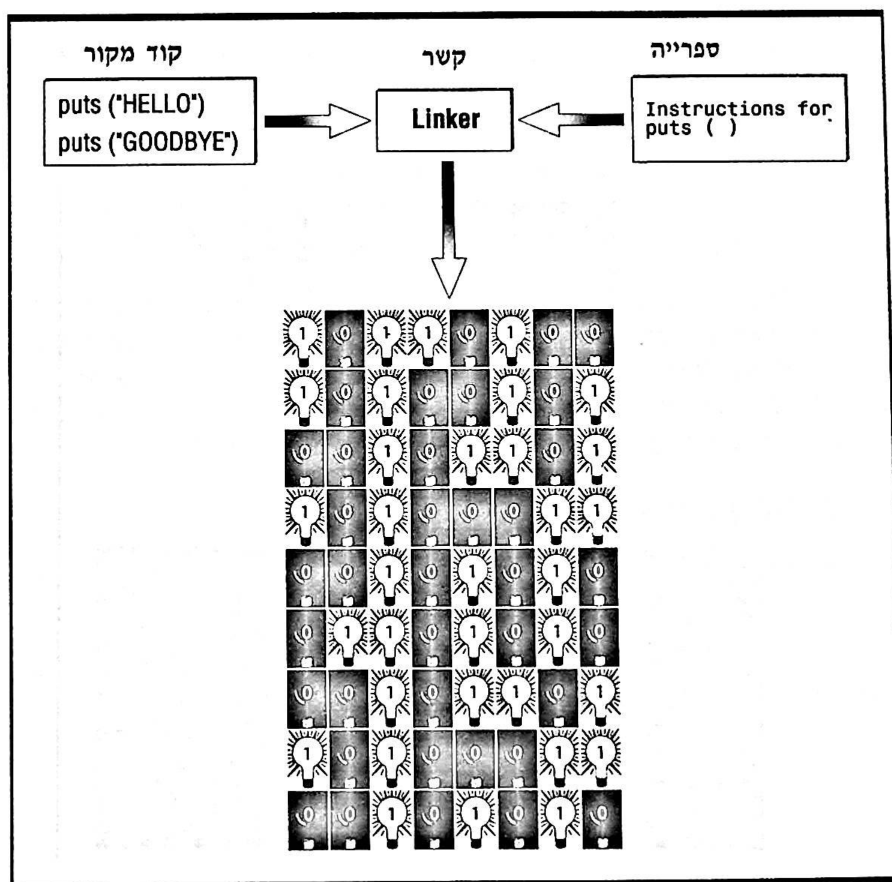
לדוגמה, C אינה כוללת פקודה להצגת מידע על-גבי הצג. עם זאת, זוהי מטלה כה שכיחה, שספריית C מכילה מספר פונקציות לפלט של מידע. במקום לכתוב את הפונקציה בעצמך, באפשרותך להשתמש באחת הפונקציות המסופקות עם הקומפיילר.

לכל פונקציה שם ייחודי, כמו הפונקציה ()puts שכבר הזכרנו קודם. במקום להקליד לתוכנית את כל ההוראות הכלולות בפונקציה, עליך רק להקליד את שם הפונקציה, תוך שימוש בתחביר ובסימני הפיסוק הדרושים. (אל דאגה, בהמשך נלמד כיצד להשתמש בשמות הפונקציות).

קובצי ספרייה מסוימים מכילים קוד מקומפל מראש. כאשר הקומפיילר נתקל בשמה של אחת מפונקציות הספרייה האלה בתוכנית, אין הוא צריך לטרוח ולהמיר את הפקודות לקוד בינארי. ההמרה כבר בוצעה. בתהליך *הקישור (linking)*, (המודגם בתרשים 1.7), מתמזגים מקטעי הקוד של הפונקציה - אותן הוראות בספרייה לביצוע הפונקציה - עם קובץ העצמים, ויוצרים תוכנית הרצה. מכיוון שהפונקציות קומפלו מראש, הקוד המקומפל יעיל מאד. יצרן הקומפיילר עידן ושיפר את הפונקציות לרמת האופטימיזציה הגבוהה ביותר.

קיימות גם פונקציות אחרות, שהשימוש בהן כה שכיח עד שקוד המקור בשפת C מסופק לך (תלוי בקומפיילר). פונקציות מסוג זה כלולות בקבצים הנקראים *קובצי כותרת*. סיומת שמם של קבצים אלה היא בדרך-כלל H. קובצי כותרת מכילים גם *הנחיות קומפיילר*, סדרת הוראות המורה לקומפיילר כיצד להשתמש בהגדרות מסוימות. במהלך הקומפילציה מתמזג הקוד שבקובץ הכותרת עם התוכנית עצמה, ליצירת קוד העצמים.

תרשים 1.8 מסכם את תהליך הקימפול-קישור.

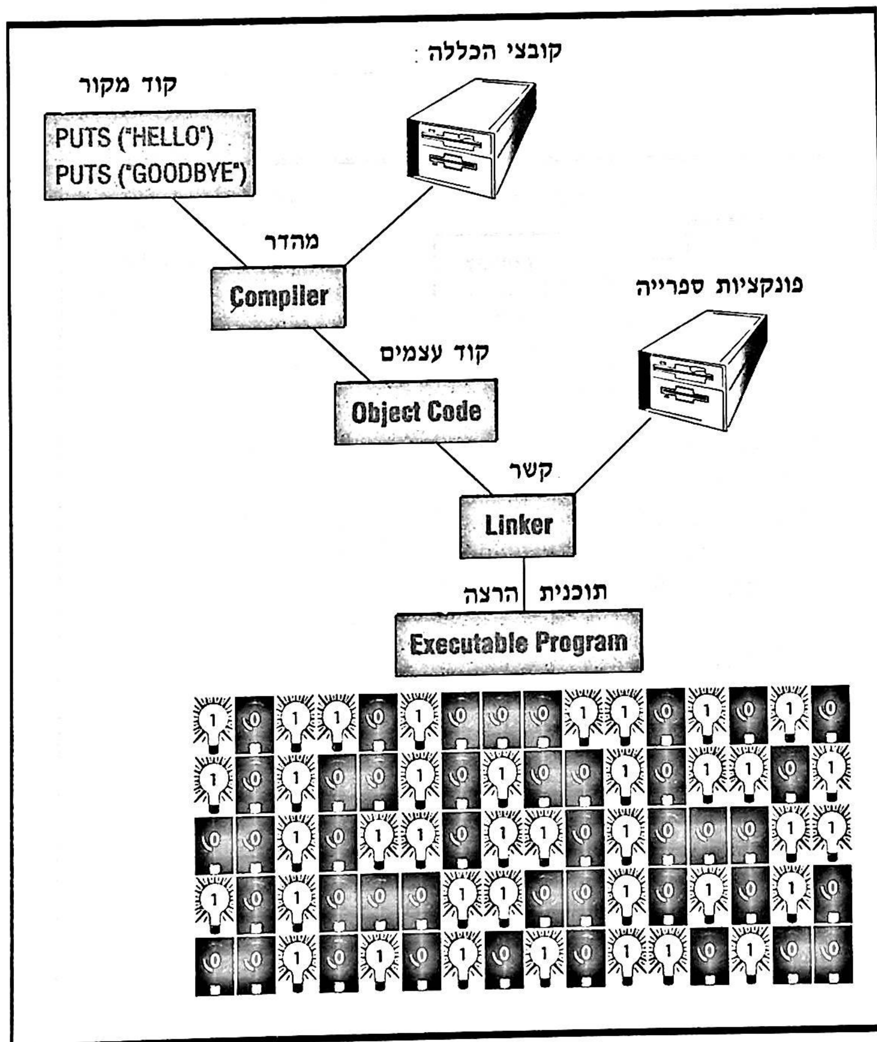


תרשים 1.7
ההוראות לביצוע הפונקציה נלקחות מהספרייה במהלך הקישור.

קובצי כותרת אינם מקומפלים. בדומה לקוד המקור בשפת C שאתה כותב בעצמך, ניתן לקרוא אותם, להדפיס אותם ולערוך אותם. עם זאת, יש להימנע מלשנות את קובצי הכותרת שסופקו עם הקומפיילר, מכיוון שבמקרה של טעות, ייתכן שהקומפיילר לא יוכל עוד ליצור קוד עצמים.

בנוסף לספרייה המסופקת עם הקומפיילר שברשותך, באפשרותך לרכוש ספריות לביצוע מטלות מיוחדות כגון ניהול מסד נתונים, גרפיקה משוכללת ומהירה, תקשורת, עבודה בחלונות ופונקציות מתקדמות אחרות. ככל שמספר הפונקציות בספריות שברשותך גדול יותר, קטנה יותר כמות העבודה שעליך לבצע בעצמך.

תרשים 1.8
תהליך הקימפול/קישור.



באפשרותך גם ליצור ספרייה משלך של פונקציות, בהן אתה משתמש בכתיבת תוכניותיך. לדוגמה, נניח שאתה כותב מספר תוכניות, שכולן מבצעות מטלות A, B ו-C. תוכל לכתוב את הפונקציות המבצעות את המטלות האלה ולהציב אותן בספרייה אישית משלך. כך הן תעמודנה לרשותך, מוכנות ומקומפלות, בכל פעם שתזדקק להן.

השימוש בקובצי ספרייה הוא שעושה את שפת C למתאימה למחשבים שונים. קומפיילר למחשב תואם-IBM מכיל קובצי ספרייה של הוראות בינאריות של מחשב IBM. קומפיילר למקינטוש כולל קובצי ספרייה של הוראות בינאריות של מקינטוש. עליך רק לכתוב את התוכנית בשפת C פעם אחת, ולהשתמש בקומפיילרים המתאימים כדי ליצור תוכניות עבור כל אחד מסוגי המחשבים.

הגרסאות השונות

שפת התכנות C פותחה ב-1972 על ידי דניס מ. ריטצ'י, והוגדרה בספר *שפת התכנות C* שחיבר ריטצ'י יחד עם בריאן וו. קרניגן. נהוג לכנות יישומים של שפת C העונים על הכללים שתוארו בספר בשם K&R C (ראשי התיבות של קרניגן וריטצ'י). הגדרת K&R נחשבת לתקן היישום המינימלי, ולכן ניתן לקמפל בהצלחה כל תוכנית הנכתבת בהתאם לכללי K&R בכל קומפיילר C.

עם זאת, קטעים אחדים בתקן לא הוגדרו במלואם. כתוצאה מכך, החלו יצרני הקומפיילרים לעדן ולהרחיב את השפה בעצמם. כדי למנוע בלבול, פיתח ב-1983 מכון התקנים הלאומי האמריקאי תקן חדש, הנקרא ANSI C. תקן זה קובע כללים להרחבת השפה, ומאמץ תקנים עבור מרבית מאפייניה של שפת התכנות C.

שפת התכנות הידועה בשם C++ היא שיפור של שפת C. אין זו שפה שונה, למעשה, מכיוון שהיא כוללת את כל הפקודות והמאפיינים של C עצמה, וכן תכונות נוספות. על ידי לימוד C תלמד גם את עיקרה של שפת C++. התוספות הכלולות בשפת C++ מאפשרות פיתוח קל יותר של תוכניות גדולות ומורכבות, באמצעות הגדרת גישה "מודולרית" יותר, בנוסף להרחבות נוספות. כמו-כן, C++ מתמקדת בתכנות מונחה-עצמים (OOP).

מהו תכנות מונחה-עצמים?

יש לומר מראש, במלוא הכנות, שאין דרך קלה ומהירה לתאר את התכנות מונחה-העצמים למי שאינם מתכנתים מנוסים. אף-על-פי-כן, ננסה לעשות זאת להלן.

דמיין לעצמך צרור של כרטיסים בגודל של 12 על 7.5 ס"מ. כל אחד מהכרטיסים מכיל שם, כתובת ופרטים נוספים של חבר במועדון כלשהו:



כרטיס חבר

שם

כתובת

מספר טלפון

מעמד

במקרה שכתובתו של אחד החברים משתנה, עליך לדפדף בכרטסת ולקרוא את השם הרשום על כל כרטיס, עד שתאתר את הכרטיס הנכון. כך גם במקרה שחל שינוי במספר הטלפון או במעמדו של אחד החברים. אם עלינו לכתוב הוראות ביצוע לשלוש הפעילויות הנפרדות האלה, הן עשויות להיראות כך:

Get the cards.

Look for Smith.

Change the address to 12 West Avenue.

Get the cards.

Look for Doe.

Change the phone number to 555-1234.

Get the cards.

Look for Jones.

Change the status to inactive.

שים לב שכל הפעילויות האלה קשורות בכרטיסים. הכרטיסים עצמם נפרדים מהפעולות שאתה מבצע עליהם. לפיכך, עליך לטפל בארבעה עצמים - הכרטיסים והפעולות של שינוי הכתובת, מספר הטלפון והמעמד.

בתכנות מונחה-עצמים אנו משלבים את הנתונים בהם אנו משתמשים (הכרטיסים) עם הפעילויות שאנו מבצעים עליהם. השילוב הזה הופך עתה לעצם היחיד בו עלינו לטפל. דמיין לעצמך את העצם (הבה נכנה אותו Member_Card) כך:

Member_Card

Name

Address

Phone Number

Status

Change_address

Change_phone number

Change_status

מכיוון שהעצם כולל הן נתונים והן פעולות, איננו צריכים עוד למסור לקומפיילר צעדים נפרדים לביצוע שינוי. הקומפיילר יבין הוראות כמו:

Member_cards.change_address(Smith, 12 West Avenue)

Member_cards.change_phone(Doe, 555-1234)

Member_cards.change_status(Jones, inactive)

ייתכן שהרעיון של תכנות מונחה-עצמים נראה לך מופשט למדי. אל דאגה, אינך צריך ללמוד תכנות מונחה-עצמים כדי לתכנת בשפת C. עם זאת, לאחר שתלמד את שפת C, תוכל לקלוט את רעיון התכנות מונחה-העצמים בקלות רבה יותר.

מה יכולה שפת C לעשות עבורך, ומה לא?

אם בכוונתך לכתוב תוכניות מחשב מסוגים שונים ובגדלים שונים - שפת C היא בחירה טובה עבורך. אין כמעט מגבלות למה שניתן להשיג באמצעות קומפיילר C חזק. ומרגע שלמדת את שפת C, המעבר לשפת ++C הוא קל ופשוט.

עם זאת, העובדה ששפת C היא בעלת עוצמה רבה אין פירושה שזוהי הדרך היחידה, או אף הטובה ביותר, לביצוע כל משימה. אם אתה זקוק ליישום של מסד נתונים, לדוגמה, אינך צריך ללמוד את שפת התכנות C. קיימים בשוק חבילות תוכנה ומחוללי יישומים המסוגלים לכתוב עבורך את היישום הדרוש. בעזרתם יש באפשרותך להציב מסד נתונים במהירות ובקלות, ולאחר מכן להוסיף לו אפיונים תוך כדי העבודה. שפת C, כשלעצמה, לא תסייע לך ליצור מסד נתונים ביום אחד. זוהי מגבלה לא רק של C, אלא של כל שפות התכנות הכלליות. שפות תכנות אלה לא נועדו לייעול הפיתוח של סוג מסוים של יישומים.

שלבים בתכנות

פיתוח תוכנית הוא תהליך לוגי, "ישר ולעניין". אם תשקיע את הזמן הדרוש לביצוע כל שלבי התהליך, מתחילתו ועד סופו, תהפוך למתכנת C מצליח. הבה נבחן את השלבים השונים של תהליך התכנות.

תכנון התוכנית

שב בנחת, לא ליד המחשב, וחשוב על מה שברצונך להשיג באמצעות התוכנית. התווה את התוכנית בפירוט רב ככל האפשר. מרבית התוכניות בנויות על-פי תבנית המכונה IPO, ראשי התיבות של Input (קלט), Processing (עיבוד) ו-Output (פלט).

לדוגמה, נניח שברצונך לכתוב תוכנית שמחשבת את סכום מס הקנייה על רכישה כלשהי. מה צריכה התוכנית לעשות?

הבה נבחן את הקלט. אנו זקוקים לשתי פיסות מידע: סכום הרכישה ושיעור מס הקנייה. אם ברצונך להשתמש בתוכנית עבור יותר מרכישה אחת, יהיה עליך להזין את סכום הרכישה בכל פעם שאתה מריץ את התוכנית. מרבית הסיכויים שתזדקק לשיעור מס אחד בלבד, זה המקובל במדינתך, ולפיכך תוכל לכלול את שיעור המס ישירות בקוד התוכנית.

עתה, יש לקבוע את תהליך עיבוד המידע. במקרה שלנו, עלינו לחשב את סכום מס הקנייה, כלומר - להכפיל את סכום הרכישה בשיעור המס.

לבסוף, נבחן את הפלט. עלינו להציג את תוצאות החישוב - הפלט - על-גבי הצג.

השלבים של תוכנית זו עשויים להיראות כך:

INPUT - קלט

אמור למשתמש שיש להזין את סכום הרכישה.

הקלד את סכום הרכישה במקלדת.

אמור למחשב מהו שיעור המס המקובל במדינה.

PROCESS - עיבוד

הכפל את סכום הרכישה בשיעור מס הקנייה.

OUTPUT - פלט

הצג את התוצאות.

כתיבת התוכנית

כדי לכתוב את התוכנית אנו משתמשים בעורך *מלל*. העורך (editor) הוא תוכנית הדומה למעבד תמלילים, אלא שהוא אינו זקוק ליכולת שיש לכל מעבד תמלילים לעבד את התצורה של התווים או של הפסקאות. למעשה, *אסור שקובץ קוד המקור יכיל קודים מיוחדים כלשהם של תצורה, מכיוון שהקומפיילר לא יבין אותם ויתייחס אליהם כאל שגיאות.*

הידור (קימפול) התוכנית

לאחר שמירת קובץ קוד המקור, השתמש בקומפיילר כדי ליצור את קובץ הביניים, הוא קובץ העצמים. במקרה שהקומפיילר נתקל בהוראות שאינו מבין, מוצגות *אזהרות קומפיילר* או *הודעות שגיאה*. אזהרה פירושה שייתכן שקיימת בעיה כלשהי, אך הקומפיילר יכול להמשיך ולייצר את קוד העצמים. הודעת שגיאה גורמת בדרך-כלל לעצירת תהליך הקימפול. במידה שמוצגות שגיאות כלשהן, עליך להטעין בחזרה את קוד המקור לעורך

המלל ולתקן את הבעיה. אלו הן בדרך-כלל *שגיאות תחביר*, טעויות באיות, בפיסוק או בסדר המלים בפקודה או בפונקציה של שפת C.

שגיאות אינן צריכות לרפות את ידיך. גם מתכנתי C מנוסים ביותר טועים לפעמים.

קישור התוכנית

אם לא דווחו שגיאות קומפילציה כלשהן, עליך לקשר את קובץ העצמים לספריות שברשותך, כדי ליצור תוכנית שניתן להריצה. אם הקשר אינו מוצא את המידע המתאים בספריות, מופיעה הודעת שגיאה. במקרה כזה, עליך לבדוק את קוד המקור ולוודא שאתה אכן משתמש בקובצי הספרייה הנכונים.

בדיקת התוכנית

כעת אתה יכול להריץ את התוכנית. אם ביצעת את כל השלבים כהלכה, התוכנית תרוץ ללא כל בעיות. עם זאת, עלולות להתגלות שגיאות משני סוגים.

שגיאת *הרצה* מתגלה כאשר התוכנית כוללת הוראה שלא ניתן לבצעה. במקרה כזה תוצג הודעה על-גבי המסך, והרצת התוכנית תיעצר. שגיאות הרצה קשורות בדרך-כלל לקבצים או להתקני חומרה.

לדוגמה, נניח שהתוכנית שלך כוללת פקודה לפתוח קובץ ששמו "ACCOUNT.DAT", אך קובץ בשם זה אינו נמצא על הכונן. הקומפיילר והקשר מניחים שהקובץ יימצא על הכונן בעת הרצת התוכנית, ולכן אינם מדווחים על שגיאה כלשהי. אולם כאשר אתה מריץ את התוכנית, אין אפשרות לבצע את ההוראה מכיוון שהקובץ אינו נמצא, והרצת התוכנית תיעצר.

שגיאות *לוגיות* מתגלות כאשר התוכנית יכולה לבצע את ההוראות, אולם ההוראות עצמן שגויות; כלומר, כשההוראות מביאות לתוצאות מוטעות. אלו הן הבעיות הקשות ביותר לאיתור, מכיוון שייתכן שאינך מודע אפילו לקיומן. עליך לבדוק בקפדנות את הפלט של התוכנית ולוודא שהתוצאות אכן מדויקות.

הבה נבחן שוב את הדוגמה של התוכנית לחישוב סכום המס. נניח ששגית והורית לתוכנית *לחלק* את סכום הרכישה בשיעור המס, במקום *להכפילו*. הקומפיילר והקשר אינם יכולים לדעת שטעית, ולכן נראה שהתוכנית נוצרה ורצה ללא תקלות. לרוע המזל, מס קנייה בשיעור של שישה אחוזים על רכישה בסכום של $100 יחושב כסכום של $1666.66! שגיאה כזו ברורה אמנם לעין, אך שגיאות לוגיות אחרות אינן כה קלות לאיתור.

אם נתקלת בשגיאות הרצה או בשגיאות לוגיות כלשהן, עליך לתקנן, ואז לקמפל ולקשר את התוכנית פעם נוספת.

לימוד תכנות

כשאתה לומד כיצד לתכנת את המחשב, אתה רוכש למעשה שני כישורים חשובים.

אתה לומד את התחביר - המלים, הדקדוק וכללי הפיסוק - של שפת המחשב עצמה. אתה לומד מה עושה כל פקודה וכל פונקציה, וכן מתי וכיצד להשתמש בהן.

אתה לומד גם את הלוגיקה של תכנות מחשבים - כיצד לבצע מטלה באמצעות שפת המחשב. זוהי הבנה אוניברסלית, אותה תוכל ליישם לכל שפת תכנות. מרגע שלמדת את הלוגיקה של התכנות בשפה אחת, תוכל לרכוש כל שפה אחרת על ידי לימוד התחביר שלה בלבד.

כדי לתכנת את המחשב יש צורך בשני הכישורים. למרבה המזל, אתה רוכש את שניהם בעת ובעונה אחת!

מה דרוש כדי לתכנת

כדי לתכנת בשפת C או בשפת C++ אתה זקוק לעורך, מהדר (קומפיילר) ולקשר.

ניתן להשתמש במעבד תמלילים רגיל, במקום בעורך מלל מיוחד, כדי ליצור את קובץ קוד המקור. אולם יש לשמור את הקובץ ללא הוראות תצורה כלשהן. צורה זו נקראת פלט ASCII או DOS TEXT. למרבית מעבדי התמלילים המקובלים יש את היכולת לשמור את הפלט כקובץ ASCII.

עם זאת, לטווח הארוך, השימוש בעורך מלל מיוחד המיועד לתכנות הוא מהיר יותר, מכיוון שהתוכנית נשמרת באופן אוטומטי ללא קודים של תצורה. עורכי מלל מסוימים מאפשרים גם להחדיר פקודות תכנות של שפת C על ידי הקלדת מלות מפתח מקוצרות, או על ידי הקשה על צירופי מקשים מיוחדים. עורך המלל הכלול בתקליטון המצורף לספר זה מציג אפילו דוגמה של תוכנית C, כדי להזכיר לך את המבנה הבסיסי של השפה.

הקומפיילר שלך יכלול גם קשר וסדרה של פונקציות ספרייה. כל ספריות C כוללות את פונקציות C הבסיסיות אותן תכיר בהמשך הספר, אולם לא כל הספריות זהות. ספריות רבות כוללות פונקציות מתקדמות המיועדות לניהול מסדי נתונים, לגרפיקה, לתקשורת ולישומים נוספים. אם אתה רוכש קומפיילר משלך, בחר קומפיילר הכולל את הפונקציות להן תזדקק.

בנוסף לפריטים הבסיסיים האלה, קיימים גם כלי-עזר נוספים היכולים לייעל את עבודת התכנות. *מאתר ומתקן התקלות (debugger)* יסייע לך לאתר שגיאות הרצה בתוכנית. הוא מציג את הערכים של המשתנים ואת שמות הפונקציות המבוצעות תוך כדי הרצת התוכנית. על ידי התבוננות בפעולה, באפשרותך לגלות היכן נפלו השגיאות. *המאפיין (profiler)* מסייע לך לבצע אופטימיזציה של התוכנית, כדי להחיש את זמן ההרצה. *המאגד (assembler)* מאפשר לך להוסיף פונקציות בשפת assembly באופן ישיר, אם ברצונך שמטלה מסוימת תורץ במהירות רבה ככל האפשר.

ישנן חבילות תוכנה הכוללות קומפיילר C יחד עם סביבת פיתוח משולבת (IDE). באמצעות IDE אתה מריץ תוכנית אחת, המעניקה לך גישה לעורך המלל, לקומפיילר, לקשר ולכלי-עזר נוספים, על ידי בחירה מתוך תפריט. אם אינך משתמש בסביבת פיתוח משולבת, עליך להפעיל את עורך-המלל, להקליד את התוכנית ולשמור את הקובץ. לאחר מכן עליך לצאת מעורך-המלל כדי להריץ את הקומפיילר.

עתידך עם שפת התכנות C/C++

הרושם הוא שהיום אין כל מגבלות למתכנתי C. זוהי השפה הנפוצה ביותר לתכנות מערכות ולפיתוח בקנה מידה רחב. שפת C איננה, כמובן, שפת התכנות היחידה העומדת היום לרשות המתכנתים, אולם נראה שהיא זוכה למרבית תשומת הלב.

לאחרונה פותחו קומפיילרים חדשים ומורחבים לשפת C, המיועדים ל-DOS, ל-Windows ולכל שאר פלטפורמות המחשוב. קיימים היום כלי-עזר לתכנות וספריות מקיפות, המייעלים את עבודת פיתוח המערכות. נראה ששפת C תשלוט בכיפה עוד זמן רב, ולכן לימוד שפת C הוא השקעה נבונה.

שאלות

1. מהו ההבדל בין מהדר (קומפיילר) לבין תרגמן (interpreter)?
2. האם כל הקומפיילרים של שפת C זהים?
3. מה ההבדל בין שפת Assembler לבין שפת מחשב עילית?
4. מהו קובץ קוד מקור?
5. מה ההבדל בין שגיאות קומפילציה לבין שגיאות הרצה?
6. מהן יתרונותיה של שפת התכנות C? הסבר.
7. מהם השלבים בתכנות מחשבים?

תרגילים

1. התווה תוכנית המחשבת תשלומי שכר רגיל ותשלומים עבור שעות נוספות, על בסיס מספר שעות העבודה בשבוע.
2. התווה תוכנית הקובעת אם אדם זכאי לגמלאות (גיל הפרישה לגמלאות הוא 65).

2

שפת התכנות C/C++

נסיון להבין מה בדיוק עושה תוכנית C מסוימת רק על ידי התבוננות בקוד המקור, עשוי להיות מרתיע בהתחלה. אומנם מרבית פקודות C הן מלים בשפה האנגלית (כמו for, למשל), ומרבית הפונקציות ב-C הן מלים וקיצורים (כמו scanf - שילוב של SCAN (סרוק) את המקלדת ו-Format (קבע את התצורה) של התווים), הרי שכאשר משלבים את הפקודות והפונקציות עם התחביר של שפת C, ועם סימני הפיסוק והרווחים, מתקבלת תוכנית שאינה נראית בדיוק כמו משהו מוכר באנגלית. אין פלא שזה מכונה "קוד". אבל אל תיתן לעובדה זו לרפות את ידיך. מרגע שתכיר את שפת C, תוכל לקרוא תוכניות C באותה קלות כמו שאתה קורא רומן טוב.

בפרק זה תלמד להכיר את המבנה של תוכנית C, וכן מספר מושגים חשובים בתכנות.

הערות ++C

לשפת C ולשפת ++C מבנה זהה. אחרי שתלמד לכתוב תוכנית בשפת C, תדע גם לכתוב תוכנית בשפת ++C.

המבנה של תוכנית C

תוכנית C מכילה פונקציה אחת או יותר. זכור, פונקציה היא סדרה של הוראות המורה למחשב לבצע מטלה מסוימת. חלק גדול מהפונקציות שתשתמש בהן כבר נכתבו וקומפלו עבורך בספריית הפונקציות שסופקה לך יחד עם הקומפיילר. במקום לכתוב את ההוראות בעצמך, עליך רק להורות לקומפיילר להשתמש באחת מהפונקציות התקניות שלו. רק במקרה שברצונך לבצע מטלה שאינה כלולה בספרייה, עליך לכתוב את הפונקציה בעצמך.

הקומפיילר המצורף לספר זה, לדוגמה, כולל ספריית פונקציות מלאה. היא מכילה את כל הפונקציות שהוגדרו בתקן K&R, את הפונקציות שהוגדרו על ידי ועדת ANSI, וכן פונקציות רבות נוספות. ניתן לקמפל ולקשר את כל התוכניות המופיעות בספר באמצעות הקומפיילר המצורף.

כל תוכנית C (וכן גם כל תוכנית ++C) מתחילה בפונקציה הנקראת ()main ונראית כך:

```
main()
```

הסוגריים המופיעים אחרי המלה ()main מהווים חלק משמה של הפונקציה וחובה לכלול אותם. אם תשכח לרשום את הסוגריים, הקומפיילר לא יוכל ליצור את התוכנית; הסוגריים מסמנים לקומפיילר שזוהי פונקציה ולא המלה main. למעשה, אחרי שמה של כל פונקציה יופיעו הסוגריים, אם כי במרבית המקרים יירשם גם משהו בתוך הסוגריים. בהמשך הספר, כל איזכור של שם פונקציה ילווה בסוגריים.

אחרי שם הפונקציה ()main מופיעות ההוראות שברצונך למסור למחשב. ההוראות יכולות לכלול פקודות C או שמות של פונקציות הכלולות בספרייה, או פונקציות שכתבת בעצמך. סוגר מסולסל פותח ({) חייב להופיע לפני ההוראה הראשונה, וסוגר מסולסל סוגר (}) חייב להופיע אחרי ההוראה האחרונה. כלומר, המבנה הבסיסי של תוכנית C נראה כך:

main()	זוהי הפונקציה העיקרית שעליך לבצע
{	והיא מתחילה כאן.
....;	
....;	אלו הן ההוראות שעליך לבצע
.....	
}	והן מסתיימות כאן.

הסוגריים המסולסלים ({ ו-}) נקראים *תוחמים* (delimiters) ומסמנים את תחילתו ואת סופו של קטע קוד. כל פונקציה שאתה כותב חייבת להיפתח ולהיסגר בסוגריים מסולסלים. בנוסף, יכולים להיות קטעי קוד בתוך הפונקציה המתחילים ומסתיימים בסוגריים מסולסלים משלהם.

כאשר אתה מפעיל את התוכנית, המחשב מתחיל לעבד את ההוראה הראשונה בפונקציה ()main. להלן תוכנית ++C/C מלאה, המציגה את המלה OK על-גבי המסך:

```
main()
{
        puts ("OK");
}
```

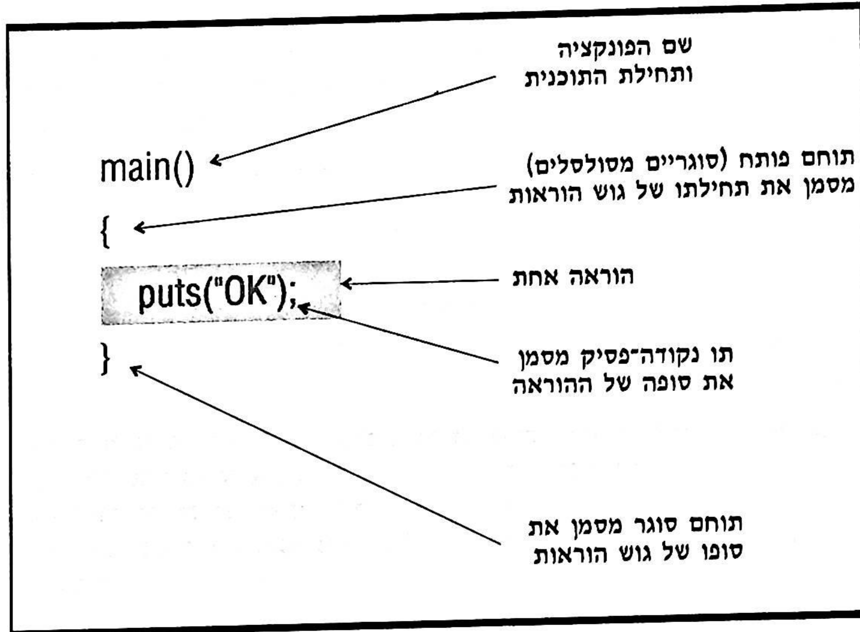
תוכנית זו כוללת הוראה אחת בלבד, אך גם עליה למלא אחר הכללים של שפת C. המרכאות סביב המלה OK לא יוצגו על-גבי המסך - תפקידן הוא רק להורות ל-C שיש להציג את התווים המופיעים בין המרכאות, ולא קבוע או משתנה ששמו OK. (בפרק 3 תלמד על קבועים ומשתנים). תרשים 2.1 מראה מה מבצע כל אחד מהחלקים של תוכנית פשוטה זו.

סימן הנקודה-פסיק (;) המופיע לאחר ההוראה נקרא *מסיים הביטוי*. סימן זה מורה לקומפיילר שהוא הגיע לסופה של הוראה - ומה שמופיע בהמשך יהיה הוראה חדשה או סופה של התוכנית. סימן הנקודה-פסיק חייב להופיע בסופה של כל הוראה:

```
main()
{
        puts("I'm OK");
        puts("You're not");
}
```

אין פירוש הדבר *שכל שורה* חייבת להסתיים בנקודה-פסיק. לעתים, הוראה מסוימת תופסת יותר משורה אחת על-גבי המסך. במקום להציב את סימן הנקודה-פסיק בסופה של השורה במסך, יש להציב את הסימן בסופה של ההוראה.

תרשים 2.1
חלקיה של תוכנית ++C/C.



דרך אגב, לאחר כל הוראת ()puts, מביא המחשב את הסמן לתחילתה של השורה הבאה. לכן, כשתריץ תוכנית המכילה שתי הוראות ()puts, יופיעו שתי שורות מלל על-גבי המסך.

שפת C ושפת ++C נקראות שפות בעלות תבנית-חופשית. אין זה משנה היכן אתה מציב את הסוגריים המסולסלים או היכן אתה מתחיל את שורות ההוראות. ניתן לכתוב את התוכנית שלנו גם באופן כזה:

```
main(){  puts("OK");}
```

כך או כך, הקומפיילר יוכל לבצע את התוכנית. אולם כדי להקל על קריאת התוכנית, אימצנו מספר מוסכמות:

- הצב את שם הפונקציה ()main בשורה נפרדת.
- הצב את התוחמים (הפותח והסוגר) בשורות נפרדות.
- רשום את ההוראות בהזחת פסקה. כשהתוכניות שלך יהפכו ארוכות יותר, תלמד כיצד מבליטות הזחות הפסקה את ההיגיון של התוכנית ומציינות את האופן בו מקובצות ההוראות ליחידות משנה.



הערה
קומפיילרים מסוימים אינם עוברים באופן אוטומטי לשורה הבאה לאחר פקודת ()puts. מידע נוסף תמצא בפרק 4.

נסה להקפיד על מוסכמות אלה ועל המוסכמות האחרות בשפת C. הן אולי לא נראות חשובות במיוחד בתוכניות קצרות, אולם הן מקלות על הטיפול בתוכניות ארוכות ומורכבות יותר.

בעוד שמספר הרווחים אינו משנה לקומפיילר, סימני הפיסוק הם משמעותיים. אם השמטת סימן תוחם, סוגריים או נקודה-פסיק, הקומפיילר ייצור שגיאה במקום תוכנית. שגיאות מסוג זה נקראות *שגיאות תחביר*, וחובה לתקן קודם שניתן יהיה ליצור את התוכנית.

אותיות גדולות, אותיות קטנות (באנגלית)

מוסכמה נוספת בשפת C נוגעת לשימוש באותיות גדולות וקטנות. פקודות ושמות פונקציות נכתבים תמיד באותיות קטנות, ולכן אנו משתמשים בצורת הכתיבה ()puts ולא ()PUTS או ()Puts. אם תשתמש באותיות גדולות לא תוצג אומנם הודעת שגיאה, אך התוכנית פשוט לא תראה כמו תוכנית C. האותיות הגדולות משמשות אך ורק עבור *קבועים* ו*שמות סמליים*.

הפונקציה ()return

מה קורה כאשר המחשב מסיים לבצע את ההוראות הכלולות בתוכנית? התוכנית נעצרת, והמחשב חוזר למצב בו היה לפני התוכנית. אם אתה מריץ את התוכנית מסימן ההנחיה של מערכת ההפעלה, יחזור ויופיע סימן ההנחיה. אם אתה מריץ את התוכנית מ-Windows, יופיע משטח העבודה של Windows.

השבת השליטה למערכת ההפעלה או ל-Windows מתבצעת באופן אוטומטי. עם זאת, קומפיילרים מסוימים של שפת C דורשים שתציין במפורש כל שלב ושלב, כולל החזרה למערכת ההפעלה. עבור קומפיילרים מסוג זה יש לכלול את הפקודה (0)return לפני התוחם הסוגר של הפונקציה ()main:

```
main()
{
    puts("I'm OK");
    puts("You're not");
    return(0);
}
```

הפונקציה ()return מורה למחשב לשוב למערכת ההפעלה. עבור מרבית הקומפיילרים הפקודה אינה הכרחית. לא תוצג הודעת שגיאה במקרה שתשמיט אותה. (גם ה-0 שבתוך הסוגריים אינו הכרחי. בהמשך תלמד מה תפקידו). עבור הקומפיילר המצורף לספר זה, לדוגמה, הפקודה ()return אינה הכרחית. אם החלטת לכלול אותה, באפשרותך לרשמה ;(0)return או אפילו ;return ללא סוגריים. עם זאת, במידה שאתה אכן כולל את *הסוגריים*, חובה להציב 0 בתוך הסוגריים - ללא ה-0 תתקבל הודעה על שגיאת קומפילציה.

אם אתה אכן כולל את הפונקציה ()return, אין להציב הוראות כלשהן בינה לבין התוחם הסוגר. לדוגמה, צורת הכתיבה להלן אינה נכונה:

```
main()
{
puts("I'm OK");
return(0);
puts("You're not");
}
```

בדוגמה זו, המחשב יחזור למערכת ההפעלה אחרי פונקצית ה-()puts הראשונה, וההוראה להציג את המלל You're not לא תתבצע כלל.

שימוש בהערות (comments)

אחרי כתיבת תוכנית C וביצוע של תיקונים ושיפורים, אין ספק שתבין היטב את התוכנית, ותדע מה עושה כל הוראה ומדוע כללת אותה בתוכנית.

אולם, ככל שהתוכניות הופכות מורכבות, קשה יותר לזכור מהי הסיבה שמאחורי כל הוראה וכל פרט בתוכנית. הדבר קשה במיוחד אם אתה חוזר ובוחן תוכנית כלשהי לאחר שחלף זמן ממושך ממועד כתיבתה, או כאשר אתה מנסה לקרוא תוכנית שנכתבה על ידי מישהו אחר.

הוספה של *הערות* עושה את התוכנית קלה יותר להבנה. הערה היא מעין מסר המופנה לכל מי שקורא את קוד המקור. הערות יכולות להופיע בכל מקום בתוכנית. הקומפיילר והקשר מתעלמים מהן. למעשה, הן אינן מתווספות לקוד העצמים או לתוכנית המוכנה להרצה.

במהלך לימוד תכנות, הוספת הערות עשויה להראות כבזבוז זמן משווע. תעדיף ודאי לעבוד על בניית התוכנית עצמה. בתוכניות מבחן קצרות ופשוטות - שאורכן שורות מעטות בלבד - ההערות אינן הכרחיות. אולם בהמשך הדרך תגלה שההערות מסייעות בהבנתה של התוכנית, אפילו תהא זו תוכנית בסיסית ביותר.

הערה מתחילה בתווים */ ומסתיימת בתווים /*, כמו בדוגמה הבאה:

```
/*This program displays a word on the monitor*/
main()
{
      puts("OK");
      return (0);
}
```

צירוף התווים */ מסמן את תחילתה של ההערה, והצירוף /* מסמן את סופה. קומפיילר C מתעלם מכל מה שמופיע בין שני הסימנים. מתכנתים נוהגים להציב בתחילתה של כל תוכנית הערה, המסבירה את ייעודה הכללי של התוכנית. הערות הנכללות בתוך גוף

התוכנית משמשות כדי להסביר הוראות מסוימות או נקודות לוגיות מיוחדות. ניתן להוסיף הוראה גם אחרי סימן הנקודה-פסיק של הוראה:

```
/*This program displays a word on the monitor*/
main()
{
     /* The word displayed is OK */
     puts("OK");
     return (0)     /*This returns to the system*/
}
```

כאשר כוללים הוראה והערה בשורה אחת, מקובל להשאיר מספר רווחים בין השתיים. הדבר מקל על קריאת ההוראה וההערה.

גם במקרה של הערות ארוכות, אין הגבלה על מספר השורות שניתן להקליד בין צירופי התווים המסמנים את תחילתה וסופה של ההערה:

```
/*This program displays a word on the monitor and includes a
return() command to make it compatible with compilers that
require it.
*/
```

צירוף התווים /* המסמן את סופה של ההערה יכול להיות בשורה נפרדת, או בשורה בה מסתיימת ההערה.

מתכנתים אחדים נוהגים להוסיף את סימן הכוכבית (*) בתחילתה של כל אחת משורות ההערה הנוספות, וקובעים את צורתה של ההערה כך:

```
/*This program displays a word on the monitor and includes a
* return() command to make it compatible with compilers that
* require it.
*/
```

באפשרותך אף להבליט את ההערה בעזרת שימוש בכוכביות נוספות:

```
/* ***
*      This program displays a word on the monitor            *
*      and includes a return() command to make it              *
*      compatible with compilers that require it.              *
*************************************************************
*/
```

הכוכביות הנוספות משמשות ל"קישוט" בלבד. הקומפיילר מתעלם מהן, כמו משאר המלל הכלול בין צירופי התווים */ ו-/*.

הערות בשפת ++C

שגיאה נפוצה אצל מתכנתים מתחילים היא להשמיט את צירוף התווים /* המסיים את ההערה. הדבר גורם להודעת שגיאה המוצגת על ידי הקומפיילר. שפת ++C עושה את השימוש בהערות לנוח מעט יותר, בכך שהיא מאפשרת להשתמש גם בצירוף התווים // לסימון הערה. הערה מסוג זה מסתיימת בסופה של השורה בה היא מופיעה, ולפיכך - אין צורך להקליד סימון מיוחד לציון סופה של ההערה:

```
// This program displays a word on the monitor
main()
{
     puts("OK");
     return  (0);    //Returns to the system
}
```

עם זאת, במקרה שההערה נמשכת על-פני יותר משורה אחת, כל שורה חייבת להתחיל בצירוף התווים //:

```
// This program displays a word on the monitor and includes a
// return() command to make it compatible with compilers that
// require it.
```

שפת ++C מתירה גם להשתמש בצירופי התווים */ ו-/* לסימון הערה.

הבנת נושא הפרמטרים

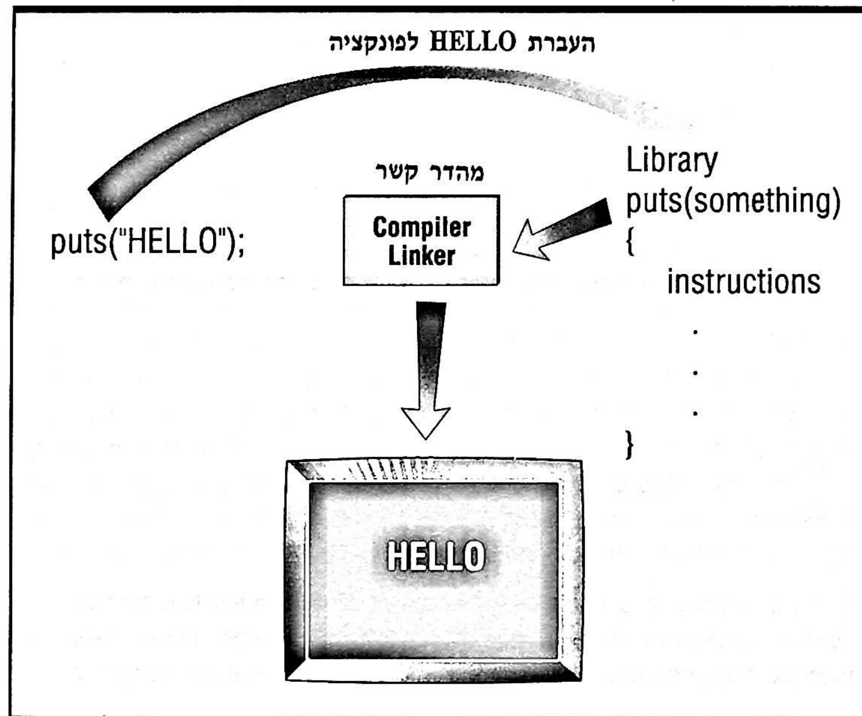
שימוש בפונקציה כלשהי, כמו ()puts, למשל, בתוך פקודה בשפת C, מכונה בשם *קריאה לפונקציה*. פירוש הדבר שאתה קורא ל-C לבצע את הפונקציה.

הסוגריים המופיעים אחרי שם הפונקציה מיועדים להכיל פרמטרים כלשהם. *פרמטר* הוא פריט מידע הדרוש לפונקציה כדי שתוכל לבצע את תפקידה. לדוגמה, ()puts היא פונקציה בספרייה. הפונקציה כוללת הוראות המורות למחשב להציג מחרוזת של תווים על-גבי המסך. אך איזה תווים עליו להציג? עליך לומר למחשב אילו תווים ברצונך להציג על ידי כך שתציב את התווים הרצויים בתוך הסוגריים. פעולה זו נקראת *העברת פרמטר*. לפיכך, בהוראה:

```
puts("HELLO");
```

המלה HELLO היא הפרמטר שאנו מעבירים לפונקציה. כפי שמודגם בתרשים 2.2, אנו מורים לקומפיילר לבצע את הפונקציה ()puts, ולהשתמש לשם כך במלה HELLO. המרכאות מסמנות שברצוננו להציג את התווים H-E-L-L-O, ולא קבוע או משתנה כלשהו (ראה פרק 3) הנקרא HELLO.

תרשים 2.2
העברת פרמטר לפונקציית ספרייה.



בספר זה אנו מכנים הוראה כמו ("puts("HELLO בשם פונקציה. למעשה, הפונקציה עצמה, ()puts, נמצאת בספרייה. לכן, מבחינה טכנית, ("puts("HELLO היא הוראה *שקוראת* לפונקציה ()puts ומעבירה לה את המלה HELLO כפרמטר.

המלה HELLO כולה מהווה פרמטר אחד. ()puts יכולה לקבל פרמטר אחד בלבד - תו, מלה או משפט, שאותו אתה רוצה להדפיס באמצעות הפונקציה. בהמשך תכיר פונקציות שיכולות לקבל מספר פרמטרים.

לפונקציות מסוימות אין צורך להעביר פרמטרים. מתכנתים רבים נוהגים לבצע את הפונקציה (0)return, למשל, באמצעות כתיבתה כך:

```
return()
```

במקרה זה אין צורך בפרמטר.

תוך כדי לימוד פונקציות C אחרות, תלמד גם פרטים נוספים אודות פרמטרים.

ודא שאתה מבין את ההבדל בין אופן השימוש בפונקציה ()main לבין כל פונקציה אחרת, כמו ()puts, למשל. ב-()main אנו משתמשים כדי להעניק שם לפונקציה המכילה את ההוראות שאנו נותנים למחשב. אנחנו לא קוראים ל-()main, אלא מבצעים את הפקודות הכלולות בה. אחת מפקודות אלה קוראת לפונקציה ()puts. לפיכך, ()puts היא פונקציה שנקראת מתוך הפונקציה ()main.

פקודה #include

אם אתה כותב תוכנית שעושה שימוש בקבצים כלשהם, או מדפיסה נתונים במדפסת, קרוב לוודאי שיהיה עליך לכלול את קובץ הכותרת stdio.h בתוך פקודה, כדלקמן:

```
#include <stdio.h>

main()

{

puts("OK");

return  (0);

}
```

הפקודה #include היא הנחיה, המורה למחשב להשתמש בנתונים הכלולים בקובץ הכותרת ששמו stdio.h. אלו הן ראשי התיבות של הביטוי standard input/output (קלט/פלט תקני), והקובץ stdio.h מכיל את כל ההוראות להן זקוק הקומפיילר כדי לעבוד עם קבצים ולשגר נתונים למדפסת.

ההוראה להשתמש בקובץ כותרת חייבת להינתן לפני ()main.

היכן נמצא קובץ הכותרת?

הקפת שמו של קובץ הכותרת בסימנים < ו-> מורה לקומפיילר שהקובץ עשוי להימצא במחיצת ברירת המחדל include. זוהי המחיצה בה מציבה תוכנית ההתקנה של הקומפיילר קובצי כותרת. אם הקובץ אינו נמצא במחיצה הנוכחית במהלך תהליך הקימפול, הקומפיילר יחפש את הקובץ במחיצה include. ניתן גם להקיף את שמו של קובץ הכותרת במרכאות, כך:

```
#include "stdio.h"
```

אך במקרה כזה, הקומפיילר יחפש את קובץ הכותרת אך ורק במחיצה הנוכחית, ואם הקובץ אינו נמצא שם, תוצג הודעת שגיאה.

תוכנית ההתקנה המצורפת לספר זה מציבה את קובצי הכותרת במחיצה בה נמצא גם הקומפיילר, C:\FIRSTC, ולכן באפשרותך להשתמש הן בסימנים < ו->, והן במרכאות. הקומפיילר כולל גם קובץ כותרת ששמו MATH.H, המשמש לביצוע פעולות מתמטיות מורכבות. מידע נוסף בנושא זה תוכל למצוא בקובץ התיעוד PCC.DOC, המותקן יחד עם הקומפיילר. התיעוד המצורף לקומפיילר שלך מסביר באיזה קובץ כותרת יש להשתמש ומתי יש להשתמש בו.

הקובץ stdio.h דרוש גם לפונקציות C מסוימות, לצורך פעולתן התקינה. לדוגמה, הפונקציה getc() קולטת תו בודד מתוך מקור כלשהו אותו אתה מציין, כמו קובץ על-גבי הכונן, למשל. עם זאת, מכיוון שנתונים רבים כל-כך נמסרים באמצעות המקלדת, שפת C כוללת גם את הפונקציה getchar(). פונקציה זו מורה לקומפיילר *לקבל תו* מהמקלדת. הפונקציה עושה זאת על ידי קריאה לפונקציה getc(), תוך כדי ציון העובדה שהמקור הוא *התקן הקלט התקני*. אנו יודעים שהתקן הקלט התקני שלנו הוא המקלדת, אך כיצד יודע זאת הקומפיילר? התקן הקלט התקני מוגדר בקובץ stdio.h. לכן, כדי שניתן יהיה להשתמש בפונקציה getchar() בתוכנית, יש לכלול את הפקודה #include עבור הקובץ stdio.h. קובץ הכותרת והספרייה פועלים בצוותא כדי להשלים את הפונקציה.

אם הקומפיילר שלך כולל קובץ ששמו stdio.h, כדאי להשתמש בפקודה #include ולכלול אותו בכל תוכנית כדי להימנע משגיאות.

עיצוב התוכנית



הערה
התיעוד המצורף לקומפיילר שלך מסביר באיזה קובץ כותרת יש להשתמש, ומתי יש להשתמש בו.

תוך כדי לימוד שפת C, תפתח גם כישורים של פתרון בעיות. כישורים אלה נחוצים כדי ליישם את השפה ואת המבנה של C לצורך ביצוע מטלות. אחד מאותם כישורים הוא היכולת לחלק את הבעיה לחלקים נפרדים, שניתן לטפל בהם.

הפרדת בעיה לחלקיה היא טכניקה מקובלת בפתרון בעיות. אחרי הכל, כלום לא קל יותר לפתור בעיה קטנה מאשר להתמודד עם בעיה גדולה! כאשר אתה ניצב בפני בעיה גדולה שקשה לפותרה, חלק אותה לבעיות קטנות שקל יותר לטפל בהן. במידת הצורך, המשך לחלק את הבעיה ליחידות יותר ויותר קטנות, עד שתוכל למצוא את הפתרון לכל יחידה. ברגע שפתרת את כל הבעיות הקטנות, הבעיה הגדולה נפתרה מעצמה.

השתמש בטכניקה זו גם בעיצוב התוכנית. התחל בחלוקת המטלה הכוללת שברצונך לבצע למטלות קטנות יותר. אם נראה לך שעדיין קשה לטפל במטלה הקטנה יותר, חלק גם אותה למטלות-משנה. המשך לעשות זאת עד שתוכל לכתוב את ההוראות המבצעות את המטלה. לאחר שכתבת את כל הקטעים הנפרדים, וקישרת אותם זה לזה בתוך ()main, התוכנית מוכנה! (בפרק 7 תלמד דרך יעילה עוד יותר לכתיבת תוכנית בקטעים).

חלוקת התוכנית בצורה כזו מסייעת לך גם באיתור שגיאות. כל שעליך לעשות הוא לשאול את עצמך איזו מטלה אינה מתבצעת כיאות, ואז לבדוק את חלק התוכנית המבצע את המטלה.

תהליך זה נקרא *איתור ותיקון שגיאות*, והוא משמש אנשי מקצוע בכל התחומים לצורך פתרון בעיות. לדוגמה, כאשר אתה מכניס את מכוניתך לתיקון במוסך, המכונאי מפנה אליך סדרה של שאלות. כשאתה מבקר אצל הרופא, הוא שואל "איפה כואב לך?". תשובותיך מסייעות לו לקבוע איזו מערכת בגוף גורמת לבעיה.

שאלות

.1 מהו המבנה הכללי של תוכנית C?

.2 מהו תוחם ביטוי?

.3 האם כל הקומפיילרים של C מחייבים שימוש בפונקציה ()return?

.4 מהי המטרה של ההערות בתוכנית C?

.5 מדוע משתמשים בפרמטר כאשר קוראים לפונקציה?

.6 האם כל קריאה לפונקציה דורשת שימוש בפרמטר?

תרגילים

1. כתוב תוכנית המציגה על-גבי המסך את המלל הבא:

Welcome to my world.

Please don't rain on my parade.

2. כתוב תוכנית המציגה את שמך, כתובתך ומספר הטלפון שלך, כשהם ממורכזים על-גבי המסך.

3. הסבר מה לקוי בתוכנית הבאה:

```
main()
(
    puts("My name is Alvin);
}
```

3

משתנים וקבועים

כל תוכנית זקוקה למידע מסוג כלשהו. תוכנית שמחשבת תשלומי משכנתא צריכה לדעת את סכום ההלוואה, את שיעור הריבית ואת משך תקופת ההחזרים. תוכנית המקטלגת אוסף בולים זקוקה לתיאור של כל בול. תוכנית שיורה בפולשים מן החלל החיצון צריכה לדעת את הכיוון של כל ירייה ושל כל טורפדו פוטונים.

מידע הנמסר למחשב נקרא נתונים. אתה מזין את הנתונים למחשב; המחשב מעבד את הנתונים בהתאם להוראותיך, ומוסר לך מידע כפלט. אך לפני שניתן יהיה להזין את התוכנית בנתונים, עליך לומר לתוכנית באיזה סוג נתונים עליה לטפל.

ראשית, על C להקצות די זיכרון כדי לאחסן כל אחד מפריטי הנתונים (מבלי לבזבז זיכרון). טיפוסים שונים של נתונים זקוקים לכמויות שונות של שטח זיכרון. שנית, לא את כל הפונקציות של שפת C ניתן לבצע על כל טיפוסי הנתונים. אם תנסה למסור לתוכנית מלה כאשר היא זקוקה למספר, תתקבל שגיאת קומפילציה או שגיאת הרצה.

כאשר אתה כותב את התוכנית, עליך לפרט לאיזה טיפוס משתייך כל פריט נתונים המשמש את התוכנית, הן כקלט והן כפלט. אינך רשאי להשמיט מידע כלשהו, ואינך יכול לשנות את דעתך כשהתוכנית כבר רצה.

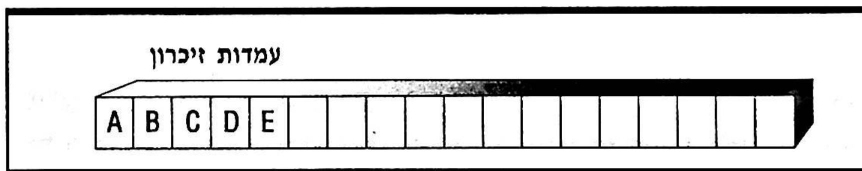
את פריטי הנתונים מסווגים על-פי סוג הערך אותו הם מכילים. (שים לב שהמושג ערך אינו מתייחס בהכרח לערך מספרי, מכיוון שנתונים יכולים להיות גם אותיות, מלים ומשפטים, בנוסף למספרים).

הערות ++C

++C כוללת את טיפוסי הנתונים הכלולים בשפת C. עם זאת, קומפיילרים מסוימים של C ושל ++C כוללים טיפוסי נתונים נוספים, שלא הוגדרו במפרט K&R המקורי של השפה.

נתונים תוויים

ערכים המשתייכים לטיפוס הנתונים תווים - המכונה בקיצור char - יכולים להיות אותיות בודדות, ספרות, או תווי מקלדת אחרים. עבור כל פריט של טיפוס הנתונים char מקצה המחשב די מקום בזיכרון לאחסון תו אחד. לכן, אם אתה עושה שימוש בחמישה פריטים שונים של טיפוס הנתונים char, המחשב יקצה מקום לחמישה תווים, כפי שמודגם בתרשים 3.1.



תרשים 3.1
כל פריט מטיפוס הנתונים char תופש עמדה אחת בזיכרון.

התווים כוללים את 26 האותיות הגדולות:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

את 26 האותיות הקטנות:

a b c d e f g h i j k l m n o p q r s t u v w x y z

את עשר הספרות:

1 2 3 4 5 6 7 8 9 0

ואת הסימנים שניתן להקליד באמצעות המקלדת:

```
! @ # $ % ^ & * ( ) _ + | \ } { ] [ " ' : ; ? / > . < ,  ~`
```

אם אתה כותב "מבחן אמריקאי", לדוגמה, באפשרותך להזין את התשובה A, B, C או D – כפריט מטיפוס הנתונים char.

כפי שתראה בהמשך הפרק, תו מטיפוס char יכול להיות גם קוד בקרה מיוחד ש-C מאחסנת באותו מקום שהיא מאחסנת תו בודד.

שים לב שפריט מטיפוס הנתונים char יכול גם להיות ספרה בודדת, כמו 1, 2 או 3. אולם C מבדילה בין התו "1" לבין המספר 1. *בתור תו*, לא ניתן להשתמש ב-"1" בפעולות מתמטיות, מכיוון שאין לו אז ערך מספרי. כשהוא משמש *מספר*, ניתן להשתמש ב-1 במתמטיקה. אך כפי שתלמד בהמשך, C יכולה לאחסן את התו "1" בחצי משטח האחסון לו היא זקוקה עבור המספר 1.

מחרוזות



הערה
קומפיילרים מסוימים של C ושל C++ כוללים טיפוסי נתונים מיוחדים למחרוזות, וכן ספרייה של פונקציות מחרוזות. הקומפיילר המצורף לספר זה כולל פונקציות לטיפול במחרוזות, אך אינו כולל טיפוסי נתונים מיוחדים למחרוזות. אם אתה משתמש בקומפיילר אחר, בדוק את התיעוד הנלווה אליו כדי לדעת איזה מאפיינים הוא כולל, וכיצד לגשת אליהם.

מחרוזת היא קבוצה של תווים, כמו מלה, ביטוי או משפט. C אינה כוללת טיפוס נתונים נפרד עבור מחרוזות, כמו בשפות תכנות אחרות. במקום זאת, אנו עובדים עם מחרוזות כסדרה של ערכים מטיפוס הנתונים char, ומשתמשים בדבר שנקרא *מערך (array)*.

מחרוזת יכולה להכיל כל צירוף של תווים, מספרים, סימני פיסוק וקודים מיוחדים המשמשים כפריטי נתונים מטיפוס char. C מבדילה בין מחרוזת המורכבת מספרות בלבד, לבין מספר. הערך של המחרוזת "123" אינו מאה עשרים ושלוש. זהו רק צירוף התווים "1", "2" ו-"3".

בספר זה תעבוד עם מחרוזות החל מהפרק הנוכחי, אם כי לא נעסוק בפירוט בנושא המערכים עד פרק 10. הסיבה לכך היא שהמחרוזות שימושיות ביותר בכל סוגי התוכניות, וניתן להתחיל להשתמש בהן גם מבלי להכיר את כל הרקע הטכני שלהן.

מספרים שלמים

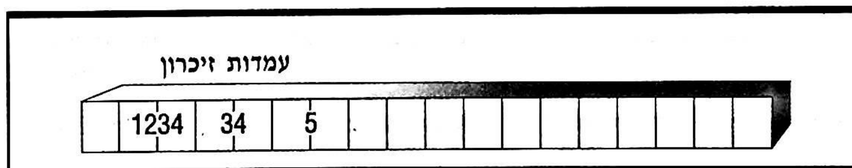
אם בכוונתך לבצע פעולות מתמטיות עם פריט נתונים, עליך להשתמש בטיפוס נתונים מספרי. C כוללת כמה טיפוסי נתונים מספריים, השונים זה מזה בתחום הערכים שניתן להקצות להם ובכמות הזיכרון שהם תופשים.

מספר שלם – הנקרא בקיצור int – הוא מספר שאינו כולל נקודה עשרונית. הוא יכול להיות חיובי או שלילי, וכן אפס, אך ללא ספרות כלשהן אחרי הנקודה.

מוסכמה מקובלת ב-C היא: "השתמש בטיפוס הנתונים int לצורך מנייה". השתמש ב-int כדי למנות כמה פעמים דבר-מה מתרחש, או כדי למנות דבר-מה המופיע ביחידות שלמות בלבד.

כפי שמודגם בתרשים 3.2, כל פריט מטיפוס הנתונים int זקוק למקום אחסון של שני תווים, בין אם זה המספר 2 או המספר 2000. אולם כדי להשתמש בשתי עמדות אחסון בלבד עבור כל מספר, C חייבת להגביל את הערך של נתוני מספרים שלמים לתחום שבין 32,768- ל-32,767. מספרים מחוץ לתחום זה זקוקים ליותר משתי עמדות אחסון.

תרשים 3.2
פריט מטיפוס הנתונים int תופש שתי עמדות אחסון.



כדי לטפל בכל סוגי המספרים, מרבית הקומפיילרים של C מגדירים כמה טיפוסים של מספרים שלמים. הערכים שניתן להקצות לטיפוסים אלה משתנים מקומפיילר אחד למשנהו.

short int	מספר שלם חיובי בין 0 ל-255
int	מספר שלם בין 32,768- ל-32,767
long int	מספר שלם בין 2,147,483,648- ל-2,147,483,647
unsigned long	מספר שלם חיובי בין 0 ל-4,294,967,295

מספרים שלמים המשתייכים לטיפוסים long int ו-unsigned long זקוקים לארבע עמדות אחסון בזיכרון. מרבית הקומפיילרים, כולל זה המצורף לספר, אינם מכירים בטיפוס הנתונים short int כטיפוס נפרד. אתה רשאי להשתמש בו בתוכנית, אולם הוא מתנהג באופן זהה לטיפוס int. הקומפיילר המצורף לספר זה מכיר בטיפוס נתונים הנקרא unsigned int בעל תחום בין 0 ל-65535.

מספרים עשרוניים

מספרים שיכולים להכיל נקודה עשרונית וספרות אחרי הנקודה נקראים ערכי *נקודה צפה*. C כוללת טיפוס נתונים מיוחד עבור מספרים עשרוניים, הנקרא float. מכיוון שמספרי נקודה-צפה יכולים להיות קטנים מאוד או גדולים מאוד, תחום המספרים מבוטא לעתים קרובות במספרים מעריכים (אקספוננציאלים). לדוגמה, ערך מטיפוס הנתונים float יכול להיות מספר גדול כמו 3.4E+38. ביטוי זה אומר: "הזז את הנקודה העשרונית 38 מקומות ימינה, תוך כדי הוספת מספר האפסים הדרוש".

קיימים טיפוסי נתונים נוספים המיועדים לטיפול בתחום רחב אף יותר של מספרים:

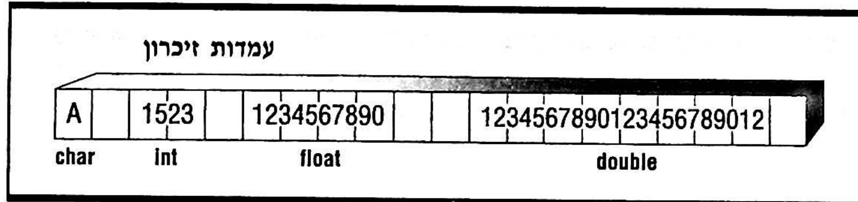
float	מספר בין 3.4E-38 ל-3.4E+38
double	מספר בין 1.7E-308 ל-1.7E+308
long double	מספר בין 3.4E-4932 ל-1.1E+4932

מידת הדיוק של טיפוסי הנתונים מסוג נקודה-צפה מוגבלת, ומשתנה מקומפיילר אחד למשנהו. לדוגמה, המספר 6.12345678912345 הוא אומנם בתחום התקף של טיפוס נתונים מסוג float, אולם מערכת המחשב שלך עשויה לאחסנו כ-6.12345 בלבד. טיפוס float כזה נחשב לבעל *דיוק יחיד*. פירוש הדבר שהוא מוגבל ל-5 או 6 ספרות אחרי הנקודה העשרונית. טיפוס double נחשב לבעל *דיוק כפול*, והוא מדויק עד ל-14 או 15 ספרות אחרי הנקודה. הקומפיילר המצורף לספר זה כולל טיפוס float בעל דיוק של שבע ספרות אחרי הנקודה, וטיפוס double בעל דיוק של 13 ספרות אחרי הנקודה. קומפיילר זה אינו תומך בטיפוס הנתונים long double.

טיפוסי נתונים מסוג float תופשים ארבע עמדות זיכרון. טיפוסי נתונים מסוג double משתמשים בשמונה עמדות זיכרון, ואילו טיפוסי long double תופשים עשר עמדות זיכרון.

תרשים 3.3 מסכם את דרישות הזיכרון של טיפוסי נתונים מקובלים בשפת C.

תרשים 3.3
סיכום דרישות זיכרון.



מדוע להשתמש במספרים שלמים?

מחוץ לעולם תכנות המחשבים, איננו מוטרדים בדרך-כלל מההבחנה שבין מספרים שלמים למספרים עשרוניים. אם אתה משתמש במחשבון לביצוע פעולות מתמטיות, כל שעליך לעשות הוא ללחוץ על הכפתורים. אם מספר מסוים כולל ספרות אחרי הנקודה, אתה מקיש אותן, אם המספר אינו מכיל ספרות אחרי הנקודה, אתה מקיש את המספר השלם. אם כך, מדוע טורחת שפת C לעשות הבחנה בין סוגי המספרים? הרי מספר הוא רק מספר!

חלק מהתשובה לשאלות אלה קשור בנושא אחסון המידע במחשב. זיכרון עולה כסף, וכאשר אין די זיכרון במחשב התוכנית לא תפעל. מתכנת טוב מנסה תמיד לחסוך בזיכרון ולעשות שימוש בכמות זיכרון קטנה ככל האפשר. שימוש במספרים שלמים במקום בערכי float, או בטיפוס התווים char במקום במחרוזות, חוסך זיכרון.

בנוסף, פעולות על מספרים שלמים מתבצעות מהר יותר מפעולות על מספרים עשרוניים. אם התוכנית שלך מבצעת כמות גדולה של פעולות מתמטיות, שימוש במספרים שלמים, היכן שניתן, הוא החלטה נבונה. ייתכן שלא תבחין בהבדל בתוכניות קטנות הרצות על מחשב מהיר, אולם ביישומים הנדסיים וגרפיים, הבדלי המהירות הם משמעותיים ביותר.

תוך כדי לימוד תכנות, הקפד לבחור תמיד בטיפוס הנתונים המתאים ביותר לצרכיך המיידיים. כדאי לפתח את ההרגל להשתמש במספרים שלמים בכל מקום שהדבר אפשרי - בכל מקרה שאתה יודע שהמספר אינו זקוק לנקודה עשרונית.

קבועים ומשתנים

לאחר שבירנת לעצמך מהו טיפוס הנתונים המתאים לכל אחד מפריטי הנתונים להם אתה זקוק, עליך להחליט כיצד תזין אותם למחשב. ישנן אומנם פקודות רבות המשמשות לקלט נתונים, אך עליך לסווג תחילה כל אחד מפריטי הנתונים כקבוע או כמשתנה.

קבוע נשאר אותו הדבר לאורך התוכנית כולה. למעשה, קבוע מקבל את הערך שלו בעת כתיבת התוכנית, ולא בזמן הרצתה, וערך זה אינו יכול להשתנות, אלא אם כן אתה משנה את התוכנית. אם ערכו של פריט נתונים כלשהו ידוע לך מראש, וערך זה אינו צריך להשתנות במהלך התוכנית, השתמש בקבוע.

לדוגמה, נניח שאתה מתגורר במדינה בה שיעור מס הקנייה הוא 5 אחוזים. כאשר אתה מחשב את מחירו של מוצר, שיעור המס יחושב תמיד כ-5 אחוזים. לכן, ניתן לאחסן את הערך 0.05, ערך מטיפוס float, כקבוע.

ערכו של *משתנה*, לעומת זאת, יכול להשתנות בכל פעם שאתה מריץ את התוכנית. אתה מציב את הערך בעת שהתוכנית מורצת. במערכת של הזמנות מוצרים למשל, שיעור מס הקנייה אינו משתנה, אך סכום ההזמנה משתנה. לא כל הרכישות הן באותו הסכום. לכן, סכום ההזמנה יאוחסן כמשתנה.

מרבית התוכניות עושות שימוש הן בקבועים והן במשתנים. כאשר אתה כותב את התוכנית עליך לדעת כיצד להתייחס לכל פריט נתונים, האם כקבוע או כמשתנה. זוהי רק אחת מההחלטות שעליך לקבל בעת שאתה מתכנן את התוכנית.

מתן שמות לקבועים ולמשתנים

עליך להעניק שם לכל קבוע ולכל משתנה בהם תעשה שימוש בתוכנית. אורכו המירבי של שם של משתנה תלוי בקומפיילר. בקומפיילרים מסוימים אתה מוגבל לשמונה תווים, ואילו קומפיילרים אחרים מתירים שמות בני עד 32 תווים או יותר. במקרים מסוימים, שמות משתנים יכולים להכיל תווים נוספים, מעבר למספר התווים המשמעותיים. לדוגמה, ייתכן שבאפשרותך להשתמש בשמות בני 32 תווים, אולם רק שמונת התווים הראשונים חייבים להיות ייחודיים. מבחינת המחשב, המשתנים accountspayable ו-accountsreceivable הם זהים, מכיוון ששמונת התווים הראשונים של שמם זהים. הקומפיילר המצורף לספר זה מתיר שימוש בעד 31 תווים משמעותיים בשמות משתנים.

שמות של משתנים וקבועים יכולים להכיל אותיות גדולות ואותיות קטנות, וכן את תו הקו התחתי (_). באפשרותך להשתמש בכל צירוף של אותיות ומספרים, אולם התו הראשון בשם חייב להיות אות. מתכנתים משתמשים בתו הקו התחתי כדי להפריד בין מלים, וכך הופכים את השמות לקריאים יותר ובעלי משמעות ברורה, כמו למשל city_tax במקום citytax.

נסה לבחור שמות המתארים את אופן השימוש בנתונים. השם city_tax מובן יותר מאשר ctax, והשם amt_due אומר יותר מאשר due. הימנע משימוש בשמות כמו A ו-B, אלא אם כן אתה כותב תוכנית פשוטה במיוחד.

אין להשתמש בפקודה של C (או במלת מפתח) כשם של משתנה או קבוע. אם תעשה זאת, הקומפיילר יציג הודעת שגיאה ויפסיק את קימפול התוכנית. הרשימה הבאה מציגה את כל מלות המפתח בשפת C ובשפת ++C, כולל מלות המפתח שנכללו בהגדרת K&R המקורית של שפת C, מלות המפתח שנוספו ב-ANSI C, ואלו שנוספו בשפת ++C. ייתכן שהקומפיילר שלך כולל פקודות נוספות המשמשות אף הן כמלות מפתח. יש לבדוק את התיעוד הנלווה. אם מוצגת לך הודעת שגיאה בעת הקימפול, ואתה בטוח שההוראה נכונה, בדוק אם אינך משתמש בטעות במלת מפתח כשם של קבוע או משתנה.

להלן מלות המפתח שהוגדרו ב-K&R C:

auto	goto
break	if
case	int
char	long
continue	register
default	return
do	short
double	sizeof
else	static
entry	struct
extern	switch
float	typedef
for	union

ANSI C הוסיפה את מלות המפתח הבאות:

cost

enum

signed

void

volatile

ואילו בשפת ++C נוספו מלות המפתח:

catch	inline
cin	new
class	operator
cout	private
delete	protected
friend	

בשמות של משתנים וקבועים ישנה הבחנה בין אותיות גדולות לקטנות. אם הענקת למשתנה את השם TAX, אינך יכול להתייחס אליו בשלב מאוחר יותר בשם Tax או tax. למעשה, יכולים להיות לך שלושה משתנים שונים, ששמם TAX, Tax ו-tax, כשכל אחד מהם מכיל ערך שונה, ומשתייך לטיפוס נתונים אחר. (כמובן שדבר זה יקשה מאד על קריאת התוכנית ועל איתור ותיקון התקלות). שגיאה נפוצה אצל מתכנתים מתחילים היא לשנות את האיות או להשתמש בצירוף שונה של אותיות גדולות וקטנות באיזכור שמו של משתנה או קבוע בחלקים שונים בתוכנית. הדבר גורם לשגיאות קומפילציה או לשגיאות הרצה, שלעתים קשה לאתרן ולתקנן.

תרשים 3.4
המוסכמות בשפת C למתן שמות למשתנים ולקבועים.

Variables

city_tax	~~city tax~~	No spaces are allowed
record1	~~1record~~	Start names with a letter
rate	~~RATE~~	Use lowercase for variable names

Constants

NAME	~~name~~	Use uppercase for constant names
TAX_RATE	~~TAX RATE~~	No spaces are allowed
COUNT1	~~1COUNT~~	Start names with a letter

המוסכמה בשפת C (כפי שמודגם בתרשים 3.4) היא להשתמש באותיות קטנות בלבד בשמות של משתנים, ובאותיות גדולות בלבד בשמות של קבועים. אומנם לא תוצג לך הודעת שגיאה אם תפר את המוסכמה, אך ישנן סיבות טובות לקיים את הכלל. שמירה על

הכלל מקלה על ההבחנה בין משתנים לבין קבועים בתוכנית, והופכת את התוכנית לנוחה יותר לקריאה ולמעקב אחר מהלכה.

חובה לציין בתוכנית C את השם והטיפוס של כל משתנה ושל כל קבוע.

הכרזה על קבוע

הכרזה על קבוע פירושה למסור לקומפיילר C את שמו ואת ערכו של הקבוע. עושים זאת לפני הפונקציה ()main, באמצעות הנחיית הקומפיילר #define, שתחבירה הכללי הוא כדלקמן:

```
#define NAME VALUE
```

אין להציב נקודה-פסיק בסופה של ההנחיה ובין ההנחיר שם הקבוע וערכו חייב להופיע לפחות רווח אחד. התחביר המדויק תלוי בטיפוס הנתונים אותו מגדירים - מספרי, char או מחרוזת (עיין בתרשים 3.5).

לפני שהקומפיילר מתחיל ליצור את קוד העצמים, הוא מחליף כל איזכור של שם הקבוע בערך של הקבוע (בדומה לפעולה אוטומטית של חיפוש והחלפה במעבד תמלילים). בעיקרו של דבר, שם הקבוע אינו מומר לעולם לקוד עצמים.

תרשים 3.5
התחביר בהכרזת קבועים.

#define NUMBER 3.15	ערכים מספריים אינם נתונים במרכאות
#define NUMBER 3	אין לכלול נקודה עשרונית בערכים מטיפוס מספר שלם
#define NUMBER 3.0 #define NUMBER 0.5	ערכים מטיפוס float חייבים לכלול ספרה עשרונית אחת, לכל הפחות, משמאל ומימין לנקודה העשרונית
#define CHARACTER 'A'	ערכים מטיפוס char נתונים בין גרשיים
#define STRING "ABC"	ערכי מחרוזת נתונים בין מרכאות

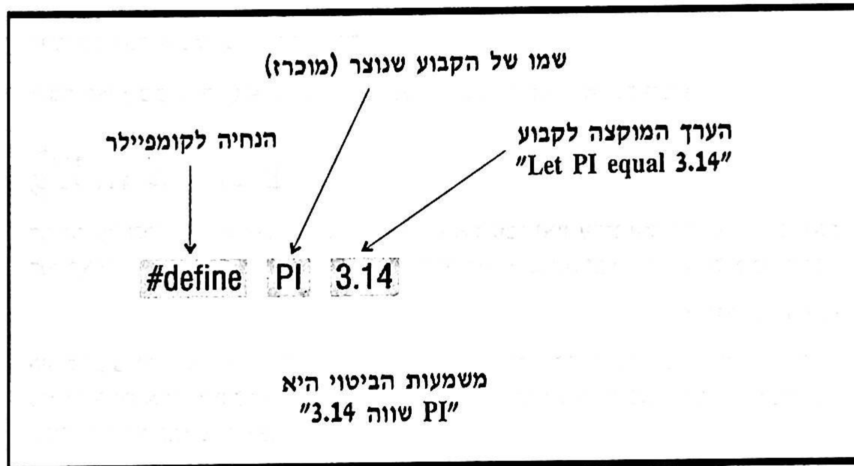
טיפ
ניתן להשתמש בהנחיה #define גם לצורך יצירת תוכניות מקרו. עיין בפרק 7.

קבוע מספרי מוכרז על ידי ציון שמו של הקבוע וערכו. לדוגמה, ההנחיה הבאה:

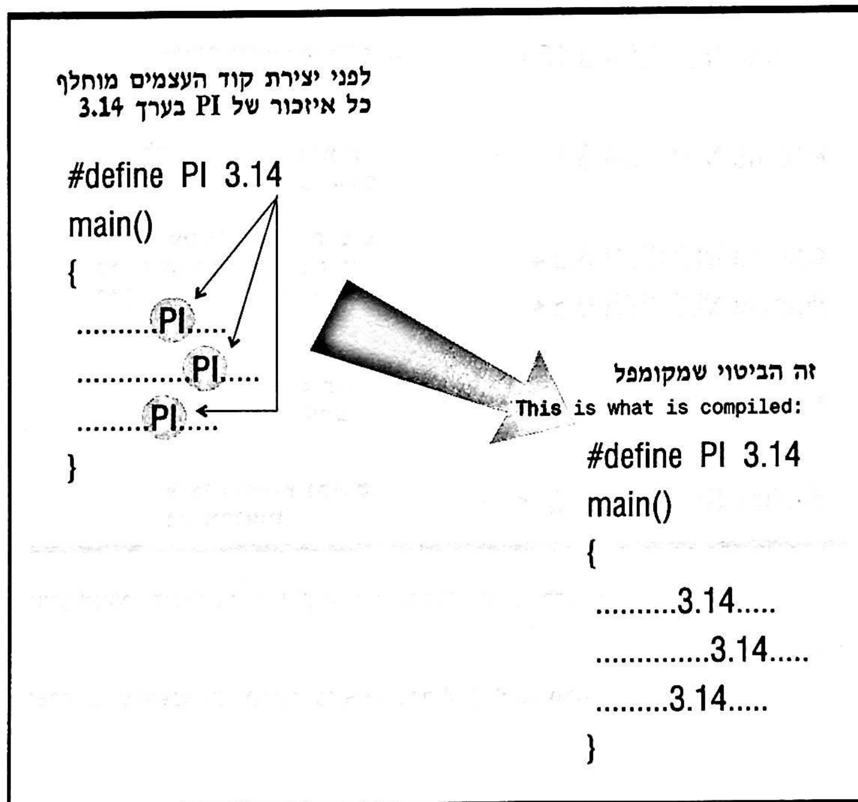
```
#define PI 3.14
```

יוצרת קבוע ששמו PI, ומציבה בו את הערך 3.14. (ראה תרשים 3.6).

תרשים 3.6
הכרזת קבוע.



תרשים 3.7
הקומפיילר מחליף את שם הקבוע בערך של הקבוע.



בכל פעם שהקומפיילר נתקל ב-PI בקוד המקור (בנוסחה למשל), הוא מציב במקומו את הערך 3.14. (ראה תרשים 3.7).

אינך חייב לציין במפורש את טיפוס הנתונים בשעה שאתה מגדיר קבוע. C מייחסת לקבוע טיפוס נתונים בהתאם לערך שמסרת בהנחיה #define. בדוגמה הקודמת, הקבוע PI מקבל את טיפוס הנתונים float, מכיוון שהערך שלו, 3.14, הוא מספר עשרוני. בהנחיה:

```
#define COUNT 5
```

הקבוע COUNT מקבל את טיפוס הנתונים int, מכיוון ש-5 הוא מספר שלם.

כדי להכריז על טיפוס float, עליך לוודא שהערך כולל לפחות ספרה אחת משמאל לנקודה העשרונית, ולפחות ספרה אחת מימין לנקודה. גם אם הערך אינו כולל ספרות כלשהן מימין לנקודה העשרונית, יש לכתוב את הנקודה וכמה אפסים מימין לה, כך:

```
#define RATE 5.00
```

ללא ציון הנקודה העשרונית, C תניח שמדובר בקבוע מטיפוס int ולא מטיפוס float. כדי להכריז על ערכים הקטנים מאחד, יש לכתוב אפס משמאל לנקודה העשרונית, כמו בדוגמה הבאה:

```
#define RATE 0.56
```

C תציג הודעה על שגיאת קומפילציה אם תכריז

```
#define RATE .56
```

בעבודה עם טיפוס הנתונים char, יש להציב גרש משני צדדיו של הערך:

```
#define INIT 'A'
```

באופן דומה, כדי להכריז על מחרוזת, יש להקיף את הערך במרכאות:

```
#define FRIEND "George"
main()
{
puts(FRIEND);
}
```

יש לזכור שהמרכאות אינן הופכות לחלק מהערך.

כאשר עושים שימוש בשם של קבוע בתור פרמטר, במקרה זה בפקודה ()puts, אין להקיף אותו במרכאות. כך יודע הקומפיילר שעליו להשתמש בערך שהוקצה לקבוע ולא בתווים המרכיבים את שמו. בדוגמה שלנו, הקומפיילר ישתמש בערך שהוצב בקבוע FRIEND, ולא בתווים F-R-I-E-N-D. אם ברצונך להציג את המלה FRIEND, הפקודה תיראה כך:

```
puts("FRIEND");
```

אחרי שהקומפיילר קרא את ההנחיה "define FRIEND "George#, בכל פעם שהוא נתקל בקבוע FRIEND בתוכנית, הוא מחליף אותו בערך "George". לפיכך, ההוראה

```
puts(FRIEND);
```

קוראת למעשה לפונקציה ()puts כך:

```
puts("George");
```

ומציגה את השם George על-גבי המסך. אם התוכנית שלך מתחילה בשורה:

```
#define FRIEND "Hazel"
```

יופיע השם Hazel על-גבי המסך. מדוע? מכיוון שההנחיה הגדירה את השם FRIEND כבעל ערך Hazel.



הערה
const היא פקודה שימושית מכיוון שהיא מאפשרת למתכנת ליצור קבועים מקומיים לפונקציות מסוימות. עם זאת, כדי להבטיח תאימות עם כל הקומפיילרים של C ושל ++C, מרבית המתכנתים ממשיכים להשתמש בהנחיה define# לצורך הכרזה על קבועים.

קבועים בשפת ++C

++C וקומפיילרים מסוימים של C כוללים דרך נוספת להכרזה על קבועים. בעזרת השימוש ב-const ניתן להכריז על קבוע, לציין את טיפוס הנתונים אליו הוא משתייך ולהקצות לו ערך. עם זאת, השימוש ב-const נעשה בתוך פונקציה, ולא כהנחיה המופיעה לפני ()main, כמו בקטע התוכנית הבא:

```
main()
{
    const int CHILDREN = 8;
    const char INIT = 'C';
```

פקודות אלה מכריזות על קבוע מטיפוס int בעל ערך של 8, ועל קבוע מטיפוס char בעל ערך של C. הן מבצעות פעולה זהה להנחיות

```
#define CHILDREN 8
#define INIT 'C'
```

מדוע להשתמש בקבועים?

אם הערך אינו עומד להשתנות בתוכנית, מדוע לטרוח ולהשתמש בקבוע? מדוע לא להשתמש בערך עצמו באופן ישיר בהוראה המתאימה בתוכנית? לדוגמה, אם אנו פותחים את התוכנית עם השורה

```
#define PHONE "555-1234"
```

ניתן להציג את מספר הטלפון על המסך באמצעות פקודה כזו:

```
puts(PHONE);
```

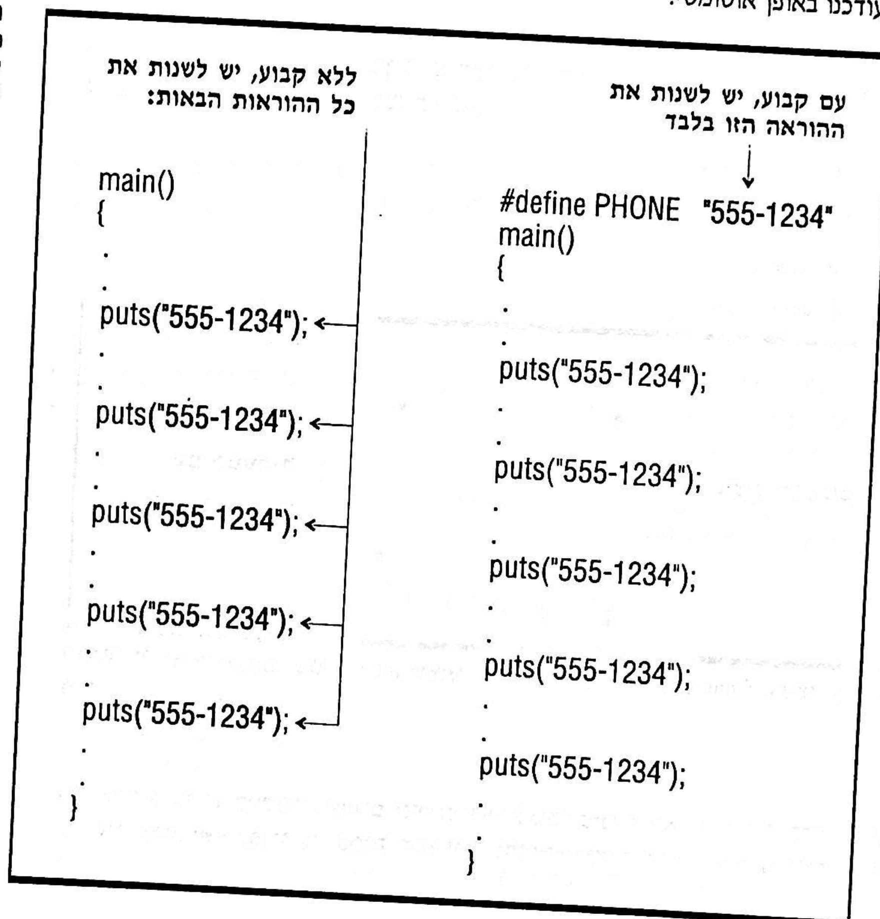
אולם מכיוון שהערך של מספר הטלפון לא משתנה במהלך התוכנית, ניתן באותה מידה להציג אותו ישירות:

```
puts("555-1234");
```

אחרי הכל, התוצאה תהיה זהה לחלוטין, ואינך צריך לטרוח ולכתוב הנחיית #define ולהעניק שם לקבוע.

יחד עם זאת, השימוש בקבועים עושה את התוכנית לקלה יותר לשינוי. נניח, לדוגמה, שהתוכנית מציגה את מספר הטלפון שלך 20 פעמים. אם חל שינוי במספר הטלפון, ולא השתמשת בקבוע, יהיה עליך לערוך מחדש את כל 20 פקודות ה-()puts. אתה עלול לטעות ולשנות 19 פקודות ()puts בלבד, ולגרום בכך שמספר הטלפון השגוי יוצג פעם אחת. אם השתמשת בקבוע, לעומת זאת, כל שעליך לעשות הוא לבצע שינוי אחד - להקליד את מספר הטלפון החדש בהנחיה #define. כפי שמודגם בתרשים 3.8, כל פקודות ה-()puts יעודכנו באופן אוטומטי.

תרשים 3.8
השתמש בקבועים כדי להקל על הכנסת שינויים בתוכנית.



הדבר נכון גם לגבי קבועים מספריים. במקום להשתמש בהנחיה

```
#define TAX 0.06
```

ולבצע חישובים תוך כדי שימוש בקבוע TAX, ניתן להציב את הערך 0.06 ישירות בנוסחאות. אך נניח שהמדינה מחליטה להעלות את שיעור המס ל-6.5%. אם לא השתמשת בקבוע, יהיה עליך לשנות את הערך בכל מקום בו הוא מופיע בתוכנית, במקום לבצע שינוי אחד בלבד בהנחיה #define.

הכרזה על משתנה

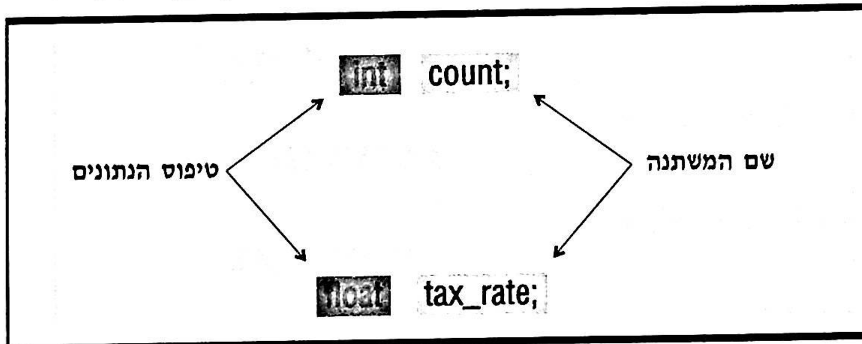
הכרזה על משתנה פירושה לספק לקומפיילר C את השם ואת הטיפוס של המשתנה, ועליך לציין במפורש את טיפוס הנתונים אליו משתייך ערכו של המשתנה. התחביר הכללי להכרזה על משתנה הוא כדלקמן:

```
type name;
```

מספר הרווחים בין הטיפוס לבין השם תלוי בך, כל עוד הוא גדול מאחד. הכרזה אופיינית על משתנה נראית כמו קטע התוכנית הבא:

```
main()
{
    int count;
    float tax_rate;
```

תרשים 3.9
הכרזה על משתנים.



הכרזה זו יוצרת משתנה מטיפוס int ששמו count, ומשתנה מטיפוס float ששמו tax_rate (ראה תרשים 3.9).

אם אתה משתמש במספר משתנים המשתייכים לאותו טיפוס, באפשרותך להכריז עליהם בהוראה אחת. יש להפריד בין שמות המשתנים בפסיק, ולסיים את ההכרזה בנקודה-פסיק, כך:

```
main()
{
    int count, children, year;
    float tax_rate, discount;
}
```

תוכנית זו יוצרת חמישה משתנים - שלושה ערכים מטיפוס הנתונים int, ושני ערכים מטיפוס float. יש להציב את הכרזת המשתנים בקבוצה בתוך הפונקציה ()main, אחרי התוחם הפותח ולפני כל הוראה אחרת. על משתנים בודדים ניתן להכריז גם לפני ()main, כך:

```
int count;
main()
{
}
```

בתוכנית פשוטה, ניתן להכריז על משתנים באחד משני המקומות. (כאשר עושים שימוש במספר פונקציות, מקום ההכרזה על משתנים נקבע על ידי כללי C מורכבים יותר. כללים אלה נדונים בפרק 7).

הקצאת ערכים

למשתנים מסוימים יש ערך *התחלתי* - הערך שיהיה למשתנה עם התחלת התוכנית. בניגוד לקבוע, ערך זה יכול להשתנות תוך כדי הרצת התוכנית. ערך התחלתי מציבים בהכרזה על המשתנה, או בהוראה נפרדת.

בעבודה עם נתונים מספריים ונתונים מטיפוס char (בהמשך נתייחס גם למחרוזות), ניתן להקצות את הערך כחלק מההכרזה:

```
main()
{
    int count=5;
    char initial='A';
    float rate=0.55;
```

הוראה זו מכריזה על משתנה ששמו count ומקצה לו את הערך ההתחלתי 5. ההוראה מכריזה גם על משתנה מטיפוס char ששמו initial ומקצה לו את האות A, ועל משתנה מטיפוס float בעל ערך התחלתי של 0.55. משני צדדיו של ערך המשתנה מטיפוס char יש

להציב גרש, וערכים מטיפוס float חייבים לכלול לפחות ספרה אחת מצידה השמאלי של הנקודה העשרונית, ולפחות ספרה אחת מצידה הימני.

לאחר שהכרזת על משתנה, ניתן להקצות לו ערך גם בהוראה נפרדת, באמצעות הסימן =, כך:

```
count=5;
initial='A';
rate=0.55;
```

הערך שאנו מקצים כאן למשתנה הוא ערך התחלתי בלבד. מכיוון שאנו עוסקים במשתנים, הערך ההתחלתי יכול להשתנות תוך כדי הרצת התוכנית.

אם אינך יודע מהו ערכו ההתחלתי של משתנה כלשהו עד שהתוכנית מורצת, באפשרותך להקצות ערך על ידי הזנתו מהמקלדת, מקובץ, או באמצעות חישוב. נלמד שיטות אלה בהמשך.

הכרזה על משתני מחרוזת

זכור, C אינה כוללת טיפוס נתונים מיוחד למחרוזות או פונקציות מובנות המיועדות למחרוזות. למרבה המזל, C מאפשרת לנו לעבוד עם מחרוזות על ידי שימוש במערכים. בתחילת הפרק הגדרנו מחרוזת כסדרה של תווים. זוהי בדיוק מחרוזת - סדרה של משתנים מטיפוס char המאוגדים במשהו שנקרא מערך. התווים המרכיבים את המחרוזת מאוחסנים יחד, בעמדות זיכרון רציפות. משתנה שהערך שלו הוא "Hello" ייראה כפי שמודגם בתרשים 3.10. כל תו מאוחסן בעמדת זיכרון נפרדת, בדיוק כמו תו char בודד, אולם ניתן להציג את המחרוזת כולה כיחידה אחת, באמצעות פקודת ()puts אחת. הסימן \0 הוא קוד מיוחד ש-C מציבה אחרי המחרוזת. תפקידו לסמן את סופה של המחרוזת, כדי שפונקציות כדוגמת ()puts ידעו מתי עליהן לחדול מלאחזר תווים לצורך הצגתם.

עמדות זיכרון

				H	e	l	l	o	\0					

משתנה מטיפוס מחרוזת

תרשים 3.10
נתוני מחרוזת בזיכרון.

הערה
הסימן \0 נקרא מסיים אפס (null terminator) ונחשב לתו אחד בלבד.

למדת כבר כיצד להכריז על קבוע מחרוזת - באמצעות הנחיית #define והקפת התווים במרכאות, כמו בביטוי הבא:

```
#define CLIENT "Joe Public"
```

כדי להכריז על משתנה מחרוזת, אנו משתמשים בטיפוס הנתונים char ומציינים את המספר המירבי של תווים שהמשתנה יכול להכיל. התחביר הוא כדלקמן:

```
char VARNAME[N];
```

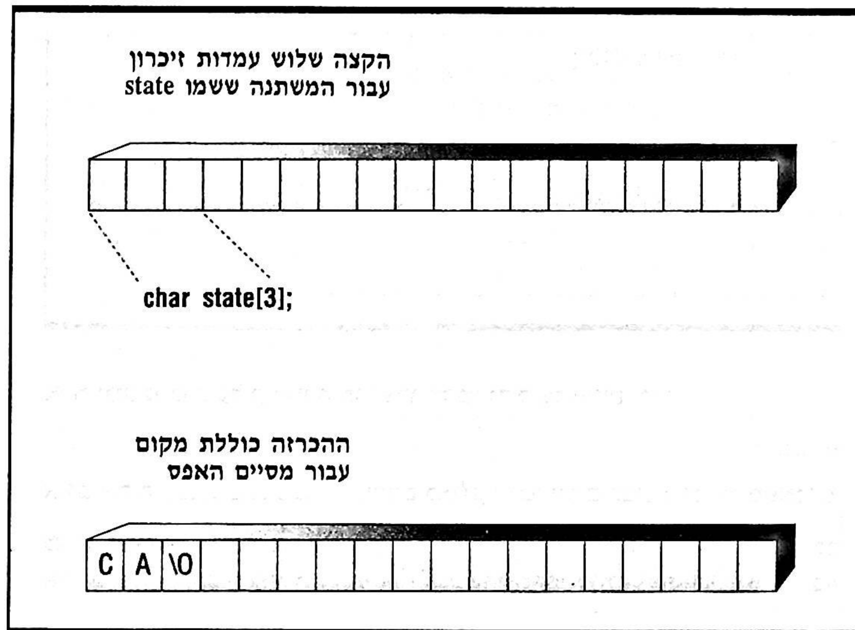
VARNAME הוא שמו של המשתנה, ו-N הוא המספר המירבי של תווים. את מספר התווים יש להציב תמיד בתוך סוגריים מרובעים, [ו-].

המספר שבתוך הסוגריים המרובעים צריך, למעשה, להיות גדול באחד מהמספר המירבי של תווים שברצונך לכלול במשתנה. C זקוקה לתו הנוסף כדי לאחסן את הקוד 0\. לדוגמה, על משתנה המיועד לאחסן קיצורים בני שתי אותיות של שמות המדינות בארה"ב ניתן להכריז כך:

```
char state[3];
```

כפי שמודגם בתרשים 3.11, הכרזה זו יוצרת משתנה מחרוזת ששמו state, היכול להכיל שני תווים בנוסף למסיים אפס (0\).

תרשים 3.11
הכרזה על משתנה מחרוזת.



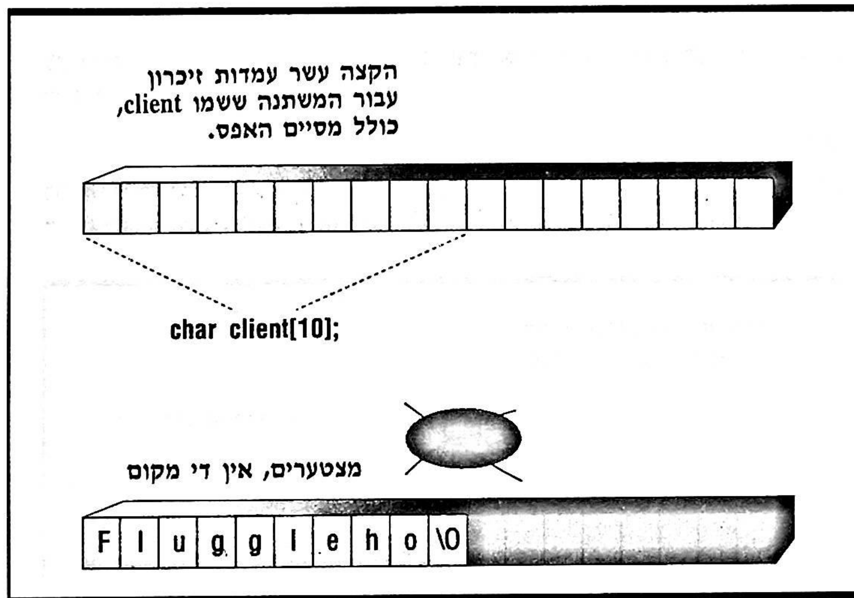
אינך יכול להקצות למשתנה מספר תווים גדול יותר מזה שהכרזת עליו. הסיבה לכך היא ש-C מקצה מקום בזיכרון המספיק רק כדי לאחסן את המספר המירבי של תווים. לכן, חשוב היטב לפני ההכרזה על המשתנה.

לדוגמה, נניח שאתה זקוק למשתנה כדי לאחסן שם של לקוח. ניתן להכריז עליו כך:

```
char client[10];
```

התוכנית עשויה לרוץ ללא תקלות, עד שתתקבל הזמנה ממר Flugglehoffen. כאשר תנסה להקצות את שם הלקוח הזה למשתנה, התוכנית תיעצר ותוצג הודעה על שגיאת הרצה, כמודגם בתרשים 3.12.

תרשים 3.12
אינך יכול להציב במשתנה תווים רבים יותר מכמות הזיכרון שהוקצתה.



אתה יכול להכריז על המשתנה תוך ציון מספר גדול של תווים, כמו

```
char client[80];
```

אולם אם התוכנית כוללת כמה משתנים כאלה, הדבר יגרום לבזבוז רב של משאבי מחשב.

לאחר שהכרזת על משתנה מחרוזת, באפשרותך להקצות למשתנה ערך. בדומה למשתנים אחרים, ניתן להקצות ערך התחלתי או לקבל ערך מהמקלדת או מקובץ. (בפרק הבא תלמד כיצד לקבל מחרוזת מהמקלדת). ב-K&R C לא ניתן להקצות ערך למחרוזת באופן ישיר, כמו שמנסה לעשות התוכנית הבאה:

```
main()
{
    char client[15];
    client="Joe";
    puts(client);
}
```

הביטוי

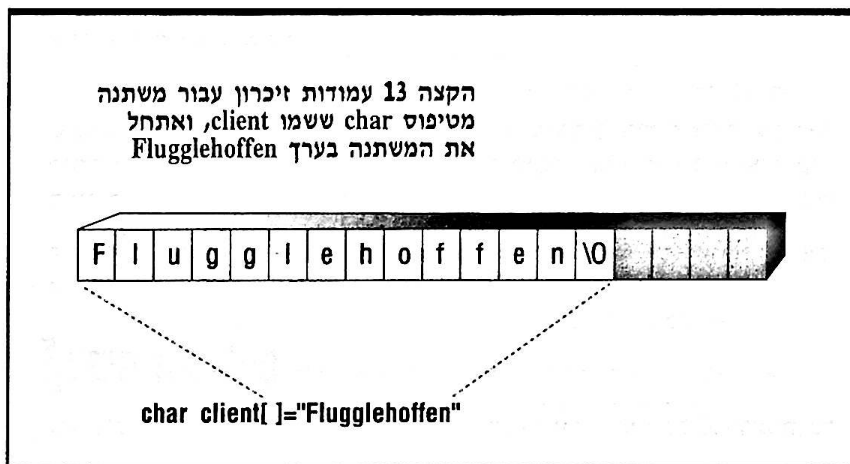
```
client="joe";
```

אינו ביטוי חוקי. אם ברצונך להציב ערך התחלתי במשתנה מחרוזת, עליך לעשות זאת בעת שאתה מכריז על המשתנים, באמצעות אחת משתי שיטות. באפשרותך להקצות את הערך על ידי הכרזה על המשתנה לפני הפונקציה ()main, כך:

```
char client[]="Flugglehoffen";
main()
{
    puts(client);
}
```

שים לב שהמספר המירבי של תווים אינו מופיע בתוך הסוגריים המרובעים. המספר המירבי המותר יהיה תלוי בקומפיילר שברשותך, אולם הוא מוגבל, בדרך-כלל, למספר המירבי של תווים בערך ההתחלתי (ראה תרשים 3.13). כמו כן, שים לב שאנחנו משתמשים בשמו של המשתנה רק בפונקציה ()puts - אינך צריך להשתמש בסוגריים מרובעים.

תרשים 3.13
הכרזה על ערך התחלתי עבור מחרוזת.



הדרך הנוספת להצבת ערך התחלתי, בתוך הפונקציה ()main, מורכבת מעט יותר:

```
main()
{
    static char greet[] = "Hello";
    puts(greet);
}
```

ההכרזה מתבצעת בתוך הפונקציה ()main, ומתחילה עם המלה static. משתנה static יכול לשמש רק בפונקציה בה הוכרז. אם ברצונך להקצות ערך התחלתי למשתנה מחרוזת ב-()main, עליך להכריז עליו כעל משתנה static. בפרק 7 נעסוק בהרחבה בנושא זה.

טיפוסי נתונים ופונקציות

טיפוס הנתונים אליו משתייכים קבוע או משתנה כלשהם קובע איזה פונקציות יכולות לעשות בהם שימוש. למרבית הפונקציות יש להעביר טיפוס נתונים מסוים בתור פרמטר. לדוגמה, ()puts דורשת מחרוזת של תווים, ולכן התוכנית

```
#define PI 3.14
main()
{
    puts(PI);
}
```

תגרום לשגיאה. הפונקציה ()puts אינה יכולה להציג ערך מטיפוס float.

שגיאה נפוצה אצל מתכנתים מתחילים היא להכריז על קבוע char על ידי הצבת ערך בן תו אחד בתוך מרכאות כפולות:

```
#define initial "A"
```

C תקבל את ההכרזה הזו, מתוך הנחה שברצונך שהקבוע יהיה מחרוזת. עם זאת, כאשר תנסה להעביר את הקבוע לפונקציה שדורשת טיפוס char, תקבל שגיאות קומפילציה. הערך הוא אומנם בן תו אחד, אבל הצבתו בתוך מרכאות כפולות מסמנת אותו כמחרוזת.

תוך כדי לימוד נושאי הקלט והפלט בפרקים הבאים, ודא שברור לך לאיזה טיפוס נתונים זקוקה כל אחת מהפונקציות.

ערכים מילוליים

ערך מילולי הוא פריט נתונים שאתה מקליד ישירות לתוך הוראה של C. לדוגמה, כל מספר, תו או מחרוזת שאתה מקליד כערך התחלתי נחשב לערך מילולי. בהוראה

```
count=5;
```

המספר 5 הוא ערך מילולי. פירוש הדבר שברצונך להקצות למשתנה את המספר הזה בדיוק. בהוראות

```
#define INIT 'C'
rate=0.55;
client="Joe";
puts("555-1234");
```

התו C, המספר 0.55, המלה Joe ומספר הטלפון 555-1234 הם ערכים מילוליים.

תכנון התוכנית

נתונים חיוניים לכל תוכנית C או ++C. אם אינך מתכנן מראש את השימוש בנתונים, רבים הסיכויים שתיתקל בשגיאות קימפול ובשגיאות הרצה.

בפרק זה עסקנו במשתנים ובקבועים מבחינת הנתונים שיש להזין לתוכנית. עליך גם להתייחס לנתונים שברצונך לקבל מהתוכנית כמידע. מרבית הסיכויים שיהיה עליך להכריז על משתנים כדי לאחסן גם את המידע הזה.

למעשה, מתכנתים רבים נוהגים להתחיל את תכנון התוכנית בפלט. בעזרת ההחלטה בדבר המידע שברצונם לקבל מהתוכנית, הם יכולים לקבוע איזה מידע יש להזין לתוכנית, וכיצד יש לעבד אותו.

כדוגמה, הבה נבחן תוכנית שמחשבת את מס הקנייה על הזמנה. להלן השלבים הכרוכים בפיתוח הקבועים והמשתנים הדרושים:

1. לאילו פריטי מידע התוכנית זקוקה:

- הפלט יהיה סכום הרכישה, כולל מס הקנייה.
- כדי לקבוע את הפלט, עליך להזין לתוכנית את סכום ההזמנה ואת שיעור מס הקנייה. סכום הרכישה מבוטא בדולרים, ולכן תזדקק למשתנה מטיפוס float. תוכל לכנות את המשתנה בשם sale. שיעור מס הקנייה נקבע על ידי המדינה, ולכן אתה זקוק לקבוע מטיפוס float. נכנה אותו TAX_RATE.
- על התוכנית להכפיל את סכום ההזמנה בשיעור מס הקנייה. לדוגמה, אם ההזמנה היא בסכום של $25.00, ושיעור מס הקנייה הוא 0.05, סכום המס הוא $1.25. תזדקק למשתנה מטיפוס float כדי לאחסן את סכום המס. נכנה את המשתנה sales_tax.
- על התוכנית להוסיף את סכום המס לסכום ההזמנה כדי לקבל את הסך-הכל. הסך-הכל בדוגמה שלנו הוא $26.25. כדי להציג את הפלט תזדקק למשתנה מטיפוס float שיאחסן את סכום הרכישה כולל המס. נקרא למשתנה זה total.

2. את הנחיות define# עבור הקבועים:

```
#define TAX_RATE 0.05
```

3. את הכרזות המשתנים:

```
float sale, sales_tax, total
```

לאחר שביצעת את התכנון עבור הנתונים, אתה מוכן לכתוב את התוכנית.

שאלות

1. מהו טיפוס הנתונים char?
2. מה ההבדל בין התו "3" לבין המספר 3?
3. באיזה טיפוס נתונים תשתמש כדי לאחסן סכום בדולרים?
4. מתי יש צורך להשתמש בתוכנית בטיפוס הנתונים long int?
5. לאיזה צורך תשתמש בטיפוס הנתונים double float?
6. מה ההבדל בין קבוע לבין משתנה?
7. כיצד מכריזים על קבוע?
8. האם C כוללת טיפוס נתונים מיוחד עבור מחרוזות?
9. האם יכול להיות למשתנה ערך אחד לאורך כל התוכנית?
10. כיצד משנים את ערכו של קבוע?

תרגילים

1. החלט איזה טיפוסי נתונים דרושים עבור תוכנית המחשבת את משכורתו השבועית של עובד, המשתכר תעריף כפול עבור כל שעת עבודה נוספת מעל 40 שעות עבודה בשבוע, וכתוב את ההכרזות המתאימות.
2. החלט איזה טיפוסי נתונים דרושים עבור תוכנית המחשבת את הסכום ואת הממוצע של ארבעה מספרים, וכתוב את ההכרזות המתאימות.
3. הסבר מה שגוי בקטע התוכנית הבא:

```
char client[3]="Ajax";
main()
float tax_due;
char name(10);
int count(5);
tax_due="$1,635.00";
```

4

הפלט בשפות התכנות C/C++

הפלט משגר עותק של נתונים מתוך זיכרון המחשב למיקום אחר - מציג אותו על-גבי המסך, מדפיס אותו במדפסת, או שומר אותו בקובץ. סוגים נדירים יותר של פלט שומרים נתונים על-גבי סרט מגנטי, מעבירים אותם דרך מודם או שולחים אותם בפקס דרך קו הטלפון.

יצירת פלט של הנתונים אינה מוחקת אותם מזיכרון המחשב, ואינה משפיעה על אופן האחסון של הנתונים. הפלט הוא רק עותק של נתונים המשוגר למקום אחר.



טיפ

כל פקודות הפלט בשפת C נתמכות גם על ידי ++C.

בפרק זה תלמד כיצד לשגר פלט אל המסך.

הפקודה בה תשתמש ליצירת הפלט תלויה בסוג הנתונים ובאופן בו ברצונך להציגם. צורות הפלט הישירות ביותר פועלות עם מחרוזות ונתונים מטיפוס char.

הפונקציה ()puts

בפרקים הקודמים הכרת כבר היטב את הפונקציה ()puts, המציגה מחרוזת על-גבי הצג. הפרמטר (המידע הנתון בסוגריים, אותו ברצונך להציג) חייב להשתייך לאחד מטיפוסי הנתונים הבאים:

- מחרוזת מילולית:

```
puts("Hello there");
```

- קבוע מחרוזת:

```
#define MESSAGE "Hello there"
main()
{
    puts(MESSAGE);
}
```



הערה

זכור, ערך מילולי הוא ערך מדויק המוקלד בתוך הוראה של C או ++C, במקום שם של משתנה או של קבוע.

- משתנה מחרוזת:

```
char greeting[]="Hello there"
main()
{
    puts(greeting);
}
```

כל טיפוס אחר של קבוע, משתנה או ערך מילולי יגרום לשגיאת קומפילציה. מחרוזת מילולית חייבת להיות נתונה במרכאות כפולות, בניגוד לשם של קבוע או של משתנה.



הערה

מכיוון שקומפיילר PCC אינו מוסיף את פקודת השורה-החדשה באופן אוטומטי, כללנו בספר זה את הקוד n\ בהוראות ()puts. אם אתה משתמש בקומפיילר שמבצע פקודת שורה-חדשה באופן אוטומטי, באפשרותך להשמיט את הקוד n\ מהוראות ה-()puts.

מרבית הקומפיילרים מבצעים פקודת שורה-חדשה אחרי הפונקציה ()puts. פירוש הדבר שאחרי הצגת הנתונים, הסמן עובר לתחילתה של השורה הבאה.

עם זאת, קומפיילרים מסוימים, ובכלל זה גם קומפיילר השותפה PCC שבתקליטון המצורף לספר זה, אינם מבצעים את פקודת השורה-החדשה באופן אוטומטי. בעבודה עם קומפיילרים אלה, עליך להורות לסמן לעבור לשורה חדשה באמצעות קוד הבקרה n\. (דיון מפורט בנושא קודים של בקרה - בהמשך הפרק). למרות שתכונה זו נראית אולי כחיסרון, היא עשויה להיות שימושית ביותר. אם אינך כולל פקודת שורה-חדשה אחרי הפונקציה ()puts, ניתן להשתמש בכמה הוראות ()puts כדי להציג נתונים בשורה אחת על המסך. השתמש בקוד הבקרה n\ רק כשברצונך שהסמן ידלג לשורת המסך הבאה.

הפונקציה ()putchar

הפונקציה ()putchar מציגה על-גבי הצג ערך char יחיד. הפרמטר חייב להשתייך לאחד מטיפוסי הנתונים הבאים:

- תו char מילולי:

```
putchar('H');
```

- קבוע char:

```
#define INITIAL 'H'
main()
{
      putchar(INITIAL);
}
```

- משתנה char:

```
main()
{
      char letter;
      letter = 'G';
      putchar(letter);
}
```

ערכו של הפרמטר מוגבל לתו אחד בלבד. ההוראה

```
putchar('H1');
```

תגרום לשגיאת קומפילציה.

אם אתה מציג תו char מילולי או קוד בקרה, יש להציבו בין גרשיים.



טיפ
הקומפיילר שברשותך עשוי לכלול את הפונקציה ()putch. פעולתה של פונקציה זו זהה לפעולתה של הפונקציה ()putchar.

מרבית הקומפיילרים של C אינם מוסיפים פקודת שורה-חדשה אחרי הוראת ()putchar. הסמן נשאר אחרי התו המוצג על המסך ואינו עובר באופן אוטומטי לשורה הבאה. כדי לדלג לשורה הבאה, עליך להשתמש בקוד הבקרה n\, בו נדון בהמשך הפרק.

פיצול אישיות

במערכות מחשב מסוימות יש לכלול את קובץ הכותרת stdio כדי להשתמש בפונקציה ()putchar. במערכות אלה, הפונקציה ()putchar נגזרת מפונקציה אחרת, ()putc. כפי שתלמד בפרק 11, הפונקציה ()putc משגרת את הפלט להתקן פלט מסוים, כגון כונן או

מדפסת. קובץ הכותרת stdio מכיל את המידע הדרוש כדי להשתמש בפונקציה ()putc לביצוע פעולתה של הפונקציה ()putchar.

מכיוון שהפונקציה ()putc יכולה לשגר את הפלט לקובץ שעל-גבי הדיסק, ישימים לגביה מספר כללים מיוחדים, וכללים אלה ישימים גם לפונקציה ()putchar. ישנם קודים מסוימים שחייבים להיכתב בקובץ, אך עשויים שלא להתאים לעמדת הזיכרון היחידה השמורה עבור פריט char. כדי שניתן יהיה להשתמש בקודים אלה, תוכננו מלכתחילה הפונקציות ()putc ו-()putchar לשימוש בטיפוס הנתונים int. הקומפיילר ממיר את טיפוס הנתונים int לתו. לפיכך, בעבודה עם קומפיילרים התומכים בתקן K&R C, באפשרותך לכתוב תוכנית כך:

```
main()
{
	int letter;
	letter = 'G';
	putchar(letter);
}
```

למרות שהמשתנה letter מוכרז כמשתנה מטיפוס הנתונים int, הוא מקבל תו בתור ערך התחלתי. התוכנית תקומפל ותרוץ ללא כל בעיה.

מתכנתים אחדים נשארו נאמנים לתקן K&R ומשתמשים תמיד בפונקציה ()putchar עם טיפוס הנתונים int. הברירה בידך.

קודים של בקרה

באפשרותך לשלוט על האופן בו נע הסמן על-גבי הצג ובכמה פעולות נוספות של המחשב, על ידי יצירת פלט של קודים מיוחדים, הנקראים *רצפי פסק (escape sequences)*. כל רצף מתחיל בסימן הלוכסן ההפוך (\), המזהה את התו הבא אחריו בתור *תו פסק*. התו המופיע אחרי הלוכסן ההפוך מסמל את הפונקציה שברצונך להפעיל. כאשר הקומפיילר נתקל בלוכסן ההפוך הוא אינו מציג את התו הבא אחריו, אלא מבצע את הפונקציה שאותו תו מסמל.

יש להציב את רצפי הפסק במרכאות בתוך הפונקציה ()puts, או בין גרשיים בפונקציה ()putchar. הרצף מורכב אומנם משני תווים, אך C מתייחסת לרצף כאל תו יחיד בתוך הוראת ()putchar.

קוד שורה-חדשה (new-line)

הרצף n\ מבצע פקודת שורה-חדשה, כלומר - מעביר את הסמן לתחילתה של השורה הבאה. השתמש בפקודה זו כדי לעבור לשורה הבאה אחרי ש-()putchar מציגה תו. אם ההוראות שלך הן:

```
putchar('A');
putchar('\n');
```

תוצג על המסך האות A, והסמן יעבור לתחילתה של השורה הבאה. שים לב שהקוד n\, כמו כל ערך מילולי אחר בפונקציה ()putchar, נתון בתוך גרשיים. מכיוון שהאות n מופיעה אחרי תו הלוכסן ההפוך, הקומפיילר מבצע את פקודת השורה-החדשה, במקום להציג את האות n על המסך.

בפקודת ()puts ניתן לשלב את הקוד עם ערך מילולי בתוך אותן מרכאות. ההוראה

```
puts("William Watson\n");
```

מציגה את השם William Watson על המסך, ומציבה את הסמן בשורה הבאה. קוד השורה-החדשה חייב להיות בתוך המרכאות של הפרמטר. אם הקומפיילר שברשותך מוסיף קוד שורה-חדשה באופן אוטומטי אחרי הפונקציה ()puts, הסמן ידלג שתי שורות למטה. התהליך פועל כך:

1. השם William Watson מוצג על-גבי המסך.
2. הקוד n\ מבצע פקודת שורה-חדשה ומעביר את הסמן לתחילתה של השורה שאחרי השורה בה מופיע השם.
3. הפונקציה ()puts מסתיימת. אם הקומפיילר שברשותך מוסיף פקודת שורה-חדשה באופן אוטומטי, הסמן מדלג שורה אחת נוספת כלפי מטה.

אומנם בדרך-כלל נעשה שימוש בקוד n\ בסופה של ההוראה, אולם אתה רשאי להציב את הקוד בכל מקום בתוך המרכאות של הפרמטר. הפקודה הבאה:

```
puts("A\nB\nC");
```

מציגה שלוש שורות מלל:

```
A
B
C
```

הקומפיילר מציג את האות A, מבצע פקודת שורה-חדשה, מציג את האות B, מבצע פקודת שורה חדשה ומציג את האות C. אם הקומפיילר שברשותך מוסיף פקודת שורה-חדשה באופן אוטומטי, הסמן יופיע בתחילתה של השורה שאחרי השורה המכילה את האות C.

קוד דילוג (Tab)

הפקודה t\ מעבירה את הסמן לנקודת הדילוג הקבועה-מראש הבאה על המסך. כדי לראות כיצד פועל קוד הדילוג, השתמש בתוכנית הבאה:

```
main()
{
    puts("1234567890123456789212345678931234567894l2345"\n);
    puts("0\t1\t2\t3\t4\t5\n");
}
```

התוכנית מציגה שורה של מספרים המיועדים לסייע לך למנות מיקום של תווים. לאחר מכן היא מציגה את המספר 0 בקצהו השמאלי של המסך, ואת המספרים 1 עד 5 בחמש עמודות הדילוג הבאות:

```
1234567890123456789212345678931234567894l2345;
0       1       2       3       4       5
```

באפשרותך להשתמש בקוד כדי ליצור עמודות של מלל או מספרים במרווחים שווים:

```
main()
{
 puts("Friends, their debts, and how long they have owed
me:\n");
 puts("Alan\t\tJeff\t\tNancy\t\tTom\n");
 puts("$1.50\t\t$2.45\t\t$6.24\t\t$3.56\n");
 puts("10 days\t\t5 days\t\t15 days\t\t1 day\n");
}
```

הפלט של תוכנית זו מוצג בתרשים 4.1. שים לב שהשתמשנו בשני קודים של דילוג בין כל שתי עמודות, כדי לרווח את העמודות על פני המסך. שים גם לב שאין רווחים נוספים בין קוד הדילוג לבין המלל הבא אחריו. כל רווח נוסף יופיע על המסך, והעמודות לא יהיו מיושרות כהלכה.

תרשים 4.1
פלט המתקבל בעזרת קודים של דילוג.

```
Friends, their debts, and how long they have owed me:

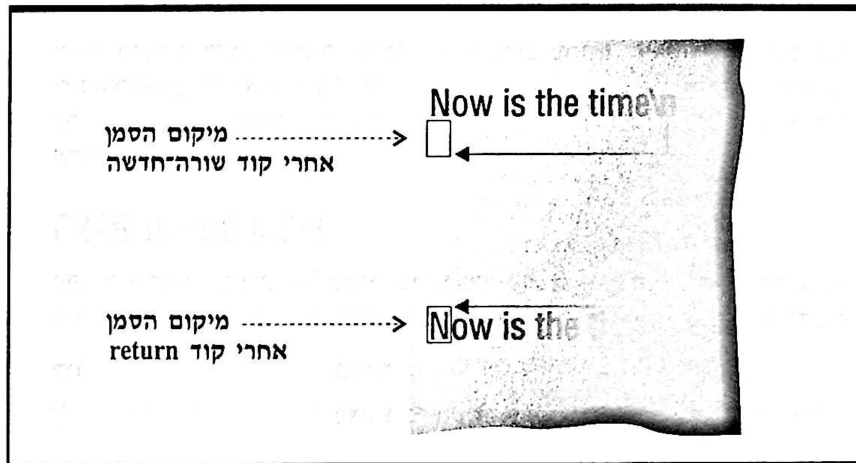
Alan            Jeff            Nancy           Tom
$1.50           $2.45           $6.24           $3.56
10 days         5 days          15 days         1 day
```

קוד Return

הקוד r\ מבצע פעולה של החזרת הסמן לתחילתה של אותה שורה, מבלי לעבור לשורה הבאה. אם אתה מציג מלל כלשהו, מחזיר את הסמן לתחילת השורה ומציג דבר-מה נוסף, המלל החדש משכתב את המלל הקודם.

תרשים 4.2 מדגים את ההבדל בין פעולת החזרת הסמן לתחילת השורה, לבין פקודת שורה-חדשה. כאשר אתה משתמש במקלדת, הקשה על מקש Enter או Return מבצעת את הפעולה המכונה בשפת C פקודת שורה-חדשה. מתכנתים אחדים מכנים את הפעולה הזו שילוב של החזרת הסמן והזנת שורה, או CR/LF. קוד return r\ בשפת C אינו מעביר את הסמן לשורה הבאה.

תרשים 4.2
ההבדל בין פקודת החזרת הסמן לבין פקודת שורה-חדשה.



את תוצאת השימוש בקוד r\ ניתן לראות בהוראה הבאה:

```
puts("Left\rRight");
```

על המסך תוצג המלה Right בלבד. הסיבה: אחרי שהקומפיילר הציג את המלה Left, החזיר קוד r\ את הסמן לתחילת השורה. עתה הוצגה המלה Right ונכתבה על-גבי המלה Left.

תנועת הסמן אל עבר תחילת השורה כתוצאה משימוש בקוד r\ אינה גורמת למחיקת המלל המוצג על המסך. הצגת תווים אחרים לפני הזזה נוספת של הסמן, לעומת זאת, גורמת למחיקת המלל הקיים.

קוד Backspace

בניגוד לקוד r\ המחזיר את הסמן לתחילת השורה, קוד Backspace, b\, מחזיר את הסמן תו אחד שמאלה. פעולה זו אינה הרסנית, כלומר - אינה גורמת למחיקת התו עליו עומד הסמן.

אם אתה משתמש בקוד backspace או בקוד return כדי להזיז את הסמן ואז נותן פקודת שורה-חדשה, הסמן עובר לשורה הבאה מבלי להשפיע על המלל המוצג על-גבי המסך. פקודת שורה-חדשה *אינה* מכניסה שורה ריקה חדשה (בניגוד לפעולתו של מקש Enter בתוכנית עיבוד תמלילים).

קוד הזנת עמוד (Formfeed)

כאשר אתה שולח פלט למדפסת (כפי שתלמד לעשות בהמשך), קוד הזנת עמוד (f\) מקדם את העמוד הנוכחי. קוד זה מוכר על ידי מרבית המדפסות.

אחרי הדפסת דו"ח, לדוגמה, מרבית המתכנתים נוהגים לכתוב קוד הזנת עמוד, כדי להבטיח שהעמוד האחרון של הדו"ח יקודם במדפסת עד סופו. אם אתה מציג את הקוד על-גבי המסך באמצעות ()puts או ()putchar, יופיע על המסך תו גרפי קטן, אולם התצוגה אינה מושפעת מעבר לכך.

הצגת תווים מיוחדים

ניתן להשתמש בקודים של בקרה כדי ליצור פלט של מגוון תווים מיוחדים. מתכנתים משתמשים לעתים קרובות ברצפי פסק כדי להציג תווים שלא ניתן להציגם בדרך אחרת:

קוד	**פעולה**
'\	מציג גרש יחיד
"\	מציג מרכאות
\\	מציג לוכסן הפוך

לדוגמה, נניח שברצונך להציג את המלל הבא:

```
We call her "Sam"
```

כולל המרכאות סביב השם Sam. אם תכתוב את פקודת ()puts כך:

```
puts("We call her "Sam"");
```

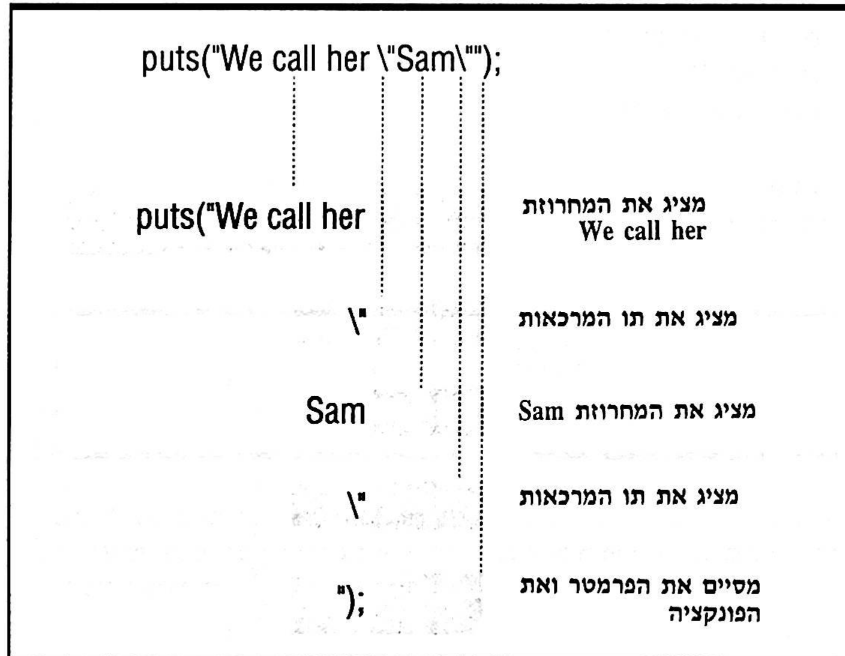
הקומפיילר יציג הודעת שגיאה. זכור, פרמטר של ()puts חייב להתחיל ולהסתיים במרכאות, כדי ש-C תדע היכן מתחילה המחרוזת והיכן היא מסתיימת. בביטוי השגוי למעלה, הקומפיילר יתרגם את הפרמטר בצורה הבאה: "We call her", ולאחר מכן התווים הנוספים ""SAM, המופיעים מחוץ למרכאות הסוגרות את הפרמטר, אך עדיין בתוך הסוגריים. מבחינת הקומפיילר, השורה פשוט מכילה מידע רב מדי.

כדי להדפיס את המלל כהלכה, השתמש בהוראה הבאה:

```
puts("We call her \"Sam\"");
```

תרשים 4.3 מראה כיצד מתרגם הקומפיילר את השורה הזו.

תרשים 4.3
יצירת פלט של תו המרכאות.



בנוסף להדפסת מרכאות ולוכסן הפוך, ניתן להשתמש בקודים של בקרה לצורך הצגת מגוון של תווים גרפיים. מחשבים אישיים של IBM ותואמי IBM יכולים להציג סדרה של תווים הידועה בשם ערכת PC (או ASCII מורחב), שכוללת את כל האותיות, המספרים וסימני הפיסוק שניתן להקליד במקלדת, וכן גם כמה סימנים גרפיים מיוחדים ואותיות יווניות. כל תו בסדרת ASCII מיוצג על ידי מספר ייחודי. לדוגמה, המספר 3 מייצג את סימן הלב. כדי להציג תו כלשהו מסדרת ASCII, יש להציב את המספר המייצג אותו אחרי תו הלוכסן ההפוך, בקוד בן שלוש ספרות, כך:

```
putchar('\003');
```

הוראה זו מציגה את תו הלב. כדי להציג את כל ארבע הצורות הקיימות בחפיסת קלפים, השתמש בתוכנית הבאה:

```
main()
{
    puts("Hearts \003\n");
    puts("Diamonds \004\n");
    puts("Clubs \005\n");
    puts("Spades \006\n");
}
```

הוראות ()puts אלה מציגות את שם הצורה ואת התו הגרפי המתאים. (תרשים 4.4 מציג את הפלט). טבלה 4.1 מראה תווים נוספים שניתן להציג בדרך זו, יחד עם מספרי קוד ASCII המתאימים.

תרשים 4.4
פלט של תווים גרפיים.

```
Hearts      ♥
Diamonds    ♦
Clubs       ♣
Spades      ♠
```

ניתן להשתמש גם בפונקציה ()putchar כדי להציג סמלים גרפיים, על ידי ניצול הכפילות המאפיינת את משתני התווים. אם אתה מכריז על משתנה מטיפוס int, באפשרותך להקצות לו ערך מספרי, כך:

```
int count;
count = 5;
```

כעת, אם תשתמש במשתנה כפרמטר מטיפוס char בהוראת ()putchar, כך:

```
putchar(count);
```

יוצג התו הגרפי שמספר זה מייצג.

טבלה 4.1
קודים של תווים שימושיים.

תו	קוד אוקטלי	תו	קוד אוקטלי
Ω	352	☺	001
δ	353	☻	002
∞	354	♥	003
ø	355	♦	004
∈	356	♣	005
∩	357	♠	006
≡	360	~(	256
±	361	»	257
≥	362	░	260
≤	363	▒	261
⌠	364	▓	262
⌡	365	α	340
÷	366	β	341
≈	367	Γ	342
°	370	π	343
•	371	Σ	344
·	372	σ	345
√	373	μ	346
n	374	τ	347
2	375	Φ	350
		θ	351

אחד הקודים בסדרת IBM/PC אינו מציג תו אלא משמיע צלצול פעמון! רצף הפסק \007 משמיע צלצול בפעמון הפנימי של המחשב. התוכנית הבאה, לדוגמה, מצלצלת בפעמון פעמיים כדי למשוך את תשומת לבו של הצופה:

```
#define BELL '\007'    /*BELL is easier to remember*/
main()   /* than \007*/
{
     putchar(BELL);    /* Sound the bell */
     putchar(BELL);    /* Sound the bell */
     puts("Attention Shoppers!\n");
     puts("There is now a sale in sporting goods"\n);
}
```

ההנחיה #define מכריזה על קבוע ששמו BELL וערכו \007. הערך מורכב אומנם מארבע הקשות במקלדת, אך הקומפיילר מקבל אותו כקבוע בן תו אחד מטיפוס char. הפעמון יצלצל בכל פעם שתיצור פלט של הקבוע BELL בהוראת putchar().

יצירת תווים מיוחדים

אם מפעמת בך רוח הרפתקנית, להלן מידע נוסף בנוגע לתווים מיוחדים. בספרי הדרכה רבים לשימוש במחשבים ובמדפסות תוכל למצוא טבלה מלאה של סדרת הקודים IBM/PC (ASCII מורחב). הקודים מוצגים, בדרך-כלל, במופע עשרוני (בסיס 10) ובמופע הקסדצימלי (בסיס 16). ישנם אומנם קומפיילרים של C שתומכים בקודים בתצורה זו, אולם תקן K&R C המקורי ו-ANSI C משתמשים בשיטת המספרים האוקטלית (בסיס 8). כדי לשמור על תאימות עם מרבית הקומפיילרים של C, השתמש בשיטה האוקטלית (כפי שמודגם בטבלה 4.1).

סדרת הקודים IBM/PC כוללת את כל התווים והסמלים שניתן להציג על המסך, ולא רק את התווים שניתן להקליד ישירות במקלדת. בספרים מסוימים אתה עשוי למצוא טבלה המציגה תת-סדרה של סדרת IBM/PC, הנקראת סדרת ASCII. תת-סדרה זו כוללת קודים עד למספר 127 (בשיטה עשרונית), כלומר-רק התווים ותנועות הסמן (כמו backspace ו-tab) שניתן להקליד ישירות במקלדת.

הפונקציה הרב-צדדית printf()

הפונקציות puts() ו-putchar() הן אומנם פונקציות שימושיות למדי, אך יש להן מספר מגבלות. הן אינן יכולות ליצור פלט של נתונים מספריים, וכל אחת מהן יכולה להציג רק ארגומנט או פרמטר אחד. גם puts() וגם putchar() יכולות להציג דבר אחד בלבד.

שפת התכנות C ושפת C++ כוללות פונקציה רב-צדדית יותר הנקראת printf(). הפונקציה printf() יכולה להציג נתונים המשתייכים לכל טיפוסי הנתונים, וכן לפעול עם מספר ארגומנטים בפרמטר אחד. בנוסף, printf() יכולה לקבוע את התצורה או הפורמט של הצגת הנתונים.

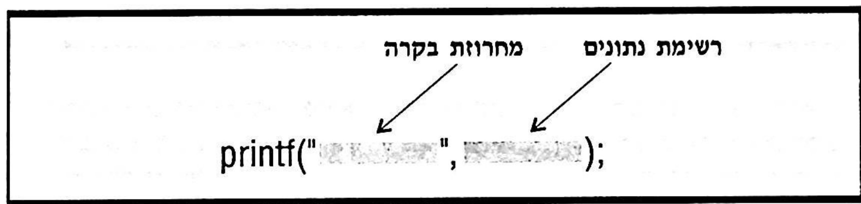
ברמה הפשוטה ביותר, ניתן להשתמש בפונקציה printf() במקום בפונקציה puts() ליצירת פלט של מחרוזת:

```
#define MESSAGE "Hello!"
main()
{
     printf(MESSAGE);
     printf(" Welcome to my world, now get out!");
}
```

בדומה לפונקציה puts(), הפונקציה printf() תציג ערכי מחרוזת מילוליים הנתונים במרכאות, וכן את הערכים של קבועי מחרוזת ומשתני מחרוזת.

הצגת מספרים

כדי ליצור פלט של נתונים מספריים ולקבוע את הפורמט של כל טיפוסי הנתונים, עליך לחלק את הפרמטר של הפונקציה printf() לשני החלקים המוצגים בתרשים 4.5.



תרשים 4.5 שני החלקים של פרמטר הפונקציה printf().

החלק הראשון נקרא *מחרוזת בקרה* או *מחרוזת פורמט*. מחרוזת הבקרה, הנתונה במרכאות, מורה לקומפיילר היכן ברצונך להציב את הנתונים בשורה המוצגת. חלק זה של הפרמטר כולל את המלל שברצונך להציג, יחד עם מצייני מיקום הנקראים *מצייני פורמט*, המסמנים את הטיפוס ואת המיקום של הנתונים.

כל מציין מתחיל בתו האחוזים (%) ולאחריו מופיעה אות המסמנת את טיפוס הנתונים:

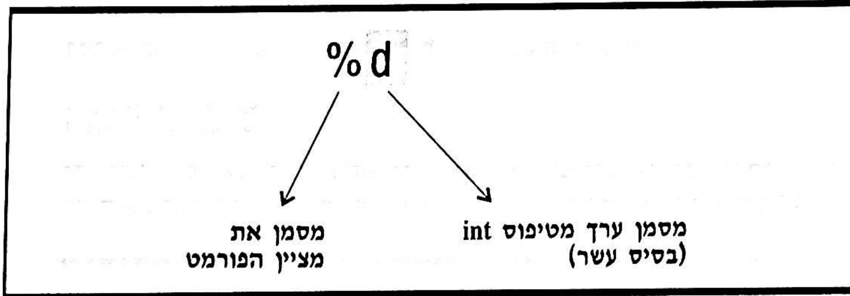
%d	מציג מספר שלם
%u	מציג מספר שלם בלתי-מסומן
%f	מציג מספר עשרוני, מטיפוס float או double
%e	מציג מספר בכתיב מדעי
%g	מציג מספר עשרוני בכתיב עשרוני או בכתיב מדעי, הקצר מביניהם
%c	מציג תו מטיפוס char
%s	מציג מחרוזת

לפיכך, חלקה הראשון של הוראת printf() עשוי להיראות כך:

```
printf("%d"
```

תו האחוזים מורה לקומפיילר שהתו הבא יהיה מציין הפורמט. (כדי להציג את תו האחוזים עצמו, יש לכתוב את התו פעמיים, כך: ;("%%")printf). התו הבא, d, מורה לקומפיילר להציג ערך של מספר שלם, בפורמט עשרוני (בסיס עשר) (ראה תרשים 4.6).

תרשים 4.6
חלקיו של מציין פורמט.



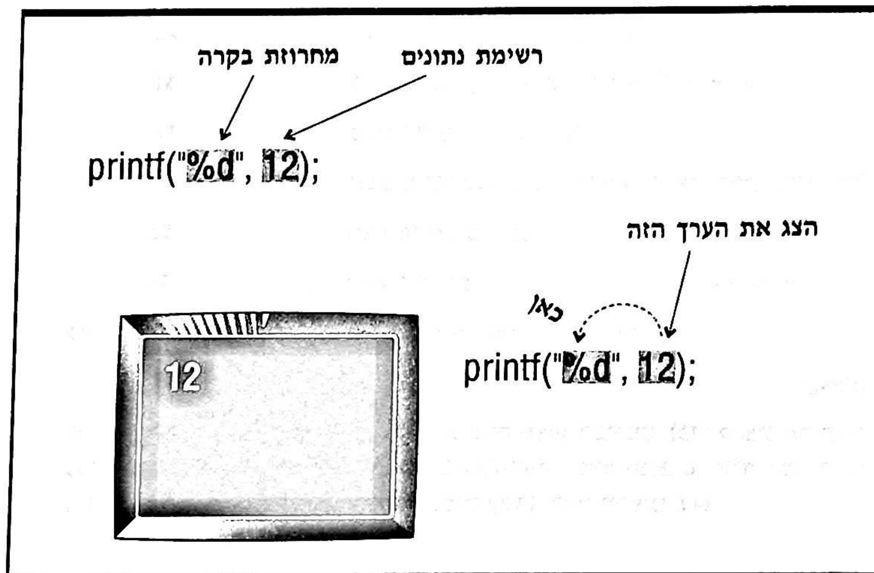
חלקו השני של הפרמטר נקרא *רשימת נתונים*, ומכיל את הערכים, הקבועים או המשתנים שברצונך להציג. יש להפריד בין רשימת הנתונים לבין מחרוזת הבקרה באמצעות פסיק, וכן להפריד בין פריטי הנתונים באמצעות פסיקים. כאשר הקומפיילר יוצר את קוד העצמים, הוא מציב את הנתונים שברשימה במקום מצייני המיקום.

הוראת ()printf מלאה, אם כי פשוטה מאד, נראית כך:

```
printf("%d", 12);
```

כשהוראות אלה יבוצעו, הערך 12 יוחדר במיקום של מציין הפורמט %d (ראה תרשים 4.7). בדוגמה זו, הפקודה ()printf מעבירה למעשה שני פרמטרים לפונקצית הספרייה - מחרוזת הבקרה וערך מילולי אחד, המספר 12.

תרשים 4.7
הערך מחליף את מציין הפורמט.



מחרוזת הבקרה יכולה להכיל גם מלל יחד עם מציין הפורמט, לצורך הצגת נתונים. כדוגמה, עיין בשורה הבאה:

```
printf("I am %d years old", 12);
```

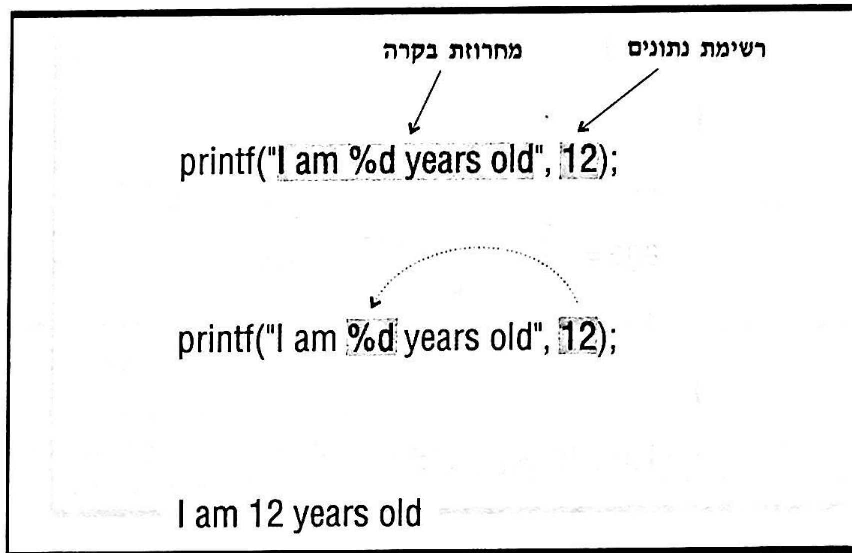
מחרוזת הבקרה כאן היא

```
"I am %d years old"
```

מציין הפורמט, %d, מראה שברצונך להציג מספר בין המלים I am לבין years old. (ראה תרשים 4.8). הקומפיילר מציב את המספר 12 במקום מציין הפורמט, כדי ליצור את הפלט:

```
I am 12 years old
```

תרשים 4.8
שימוש במציין פורמט בתוך מלל.



הוראה זו מעבירה לפונקציה הן מחרוזת מילולית והן ערך מספרי.

כמובן שניתן להשיג את התוצאה הזו על ידי הצגת השורה כולה כמחרוזת מילולית, בעזרת אחת מההוראות הבאות:

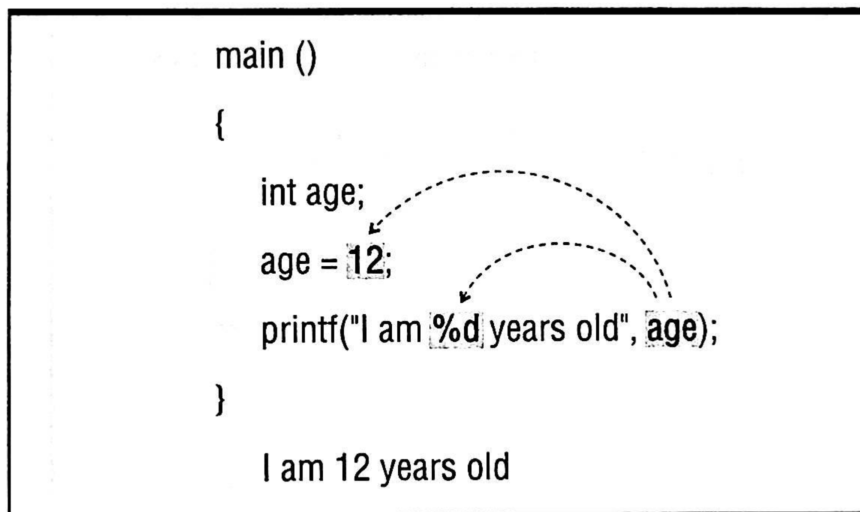
```
printf("I am 12 years old");
puts("I am 12 years old");
```

אולם כדי לשלב מלל עם קבוע או משתנה, עליך להשתמש בפונקציה printf() ובמציין פורמט, כמו בתוכנית הבאה:

```
main()
{
    int age;
    age = 12;
    printf("I am %d years old", age);
}
```

במקרה זה, התוכנית מציגה מחרוזת מילולית ומשתנה מטיפוס int בהוראה אחת (כמודגם בתרשים 4.9).

תרשים 4.9
הערך של המשתנה מחליף את מציין הפורמט.

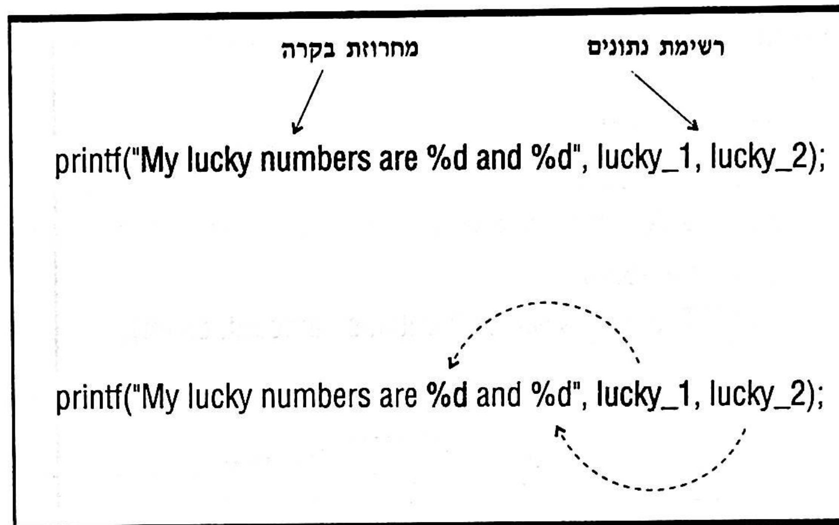


ניתן להעביר כל מספר רצוי של פרמטרים כדי להציג ארגומנטים מרובים, אולם יש לכלול מציין פורמט עבור כל ארגומנט. הערכים ברשימת הנתונים חייבים להופיע בסדר זהה לסדר הופעת המציינים המתאימים להם. הפריט הראשון ברשימה מחליף את מציין הפורמט הראשון, הפריט השני מחליף את המציין השני, וכך הלאה. עיין בתוכנית הבאה:

```
main()
{
    int lucky_1, lucky_2;
    lucky_1 = 12;
    lucky_2 = 21;
    printf("My lucky numbers are %d and %d", lucky_1, lucky_2);
}
```

התוכנית מכריזה על שני משתנים מטיפוס int, המשתנים lucky_1 ו-lucky_2, ומקצה להם ערכים התחלתיים. רשימת הנתונים בהוראת ה-()printf מכילה שני משתנים (ארגומנטים) אותם יש להציג, ולכן חייבת מחרוזת הבקרה לכלול שני מצייני פורמט. מכיוון ששני המשתנים הם מטיפוס int, חייבים שני מצייני הפורמט להיות d%, כמודגם בתרשים 4.10.

תרשים 4.10
שימוש בשני מצייני פורמט.



הקומפיילר מציב את הערכים של המשתנים במקום מצייני הפורמט, כדי להציג את הפלט הבא:

```
My lucky numbers are 12 and 21
```

ערכו של המשתנה lucky_1, הפריט הראשון ברשימת הנתונים, מוצג במיקום של המציין הראשון. הערך של המשתנה lucky_2, הפריט השני, מוצג במיקום של המציין השני. אם סדר הפריטים היה הפוך:

```
printf("My lucky numbers are %d and %d", lucky_2, lucky_1);
```

היו הערכים מוצגים בסדר הבא:

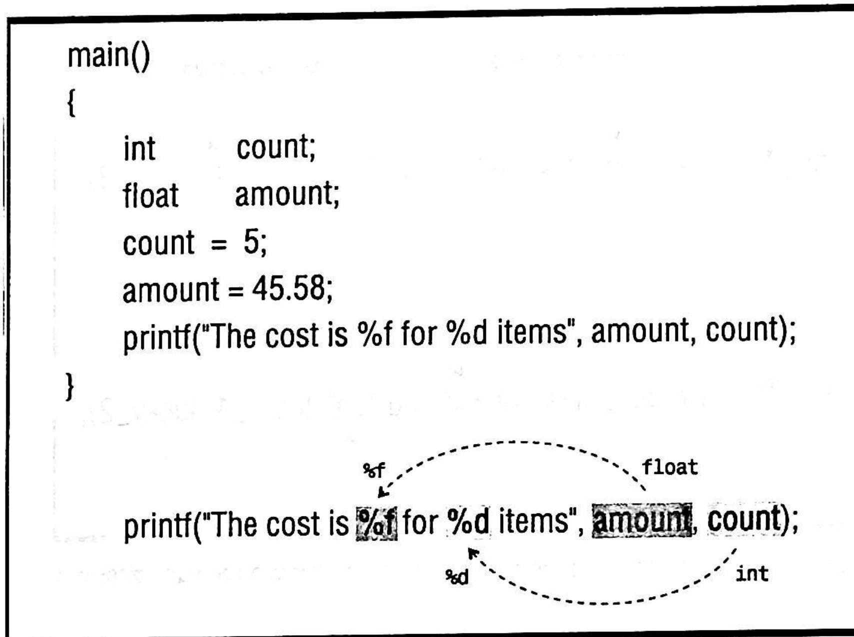
```
My lucky numbers are 21 and 12
```

טיפוס הנתונים צריך להתאים לטיפוס הנתונים עליו מצביע מציין הפורמט. לדוגמה, התוכנית הבאה מציגה משתנה מטיפוס int ומשתנה מטיפוס float בפונקציה ()printf אחת:

```
main()
{
    int count;
    float amount;
    count = 5;
    amount = 45.58;
    printf("The cost is %f for %d items", amount, count);
}
```

ערכו של הפריט הראשון ברשימת הנתונים, מטיפוס float, מוחדר במיקום של המציין הראשון, %f. ערכו של הפריט השני, מטיפוס int, מוחדר במיקום של המציין השני, %d. התוכנית מקומפלת ורצה ללא שגיאות מכיוון שטיפוסי הנתונים מתאימים, כמודגם בתרשים 4.11.

תרשים 4.11
טיפוס הנתונים חייב להתאים למציין הפורמט.



```
main()
{
    int     count;
    float   amount;
    count  =  5;
    amount = 45.58;
    printf("The cost is %f for %d items", amount, count);
}
```


ערך float מחליף את המציין %f, וערך int מחליף את המציין %d. הפלט הוא:

```
The cost is 45.580000 for 5 items
```

(מספר האפסים תלוי בקומפיילר שברשותך. בהמשך תלמד מדוע הם מופיעים). לעומת זאת, אם הוראת printf() היתה נכתבת כך:

```
printf("The cost is %f for %d items", count, amount);
```

ייתכן שהקומפיילר לא יזהה שגיאה, אולם הפלט יהיה חסר משמעות, כמו בביטוי הבא:

```
The cost is -2.002149E37 for 16454 items
```

טיפוסי הנתונים המספריים אינם מתאימים למצייני הפורמט. ניתן להציג מספר ארגומנטים מטיפוסים שונים בתוך פרמטר אחד, רק כאשר טיפוסי הנתונים מתאימים למצייני הפורמט.

הזנת שורה (Line Feed)

הפונקציה printf() אינה מוסיפה פקודת שורה-חדשה באופן אוטומטי אחרי הצגת נתונים. לאחר שהפרמטר מוצג, הסמן נשאר באותה השורה אחרי התו האחרון.

אם ברצונך שהסמן יעבור לשורה הבאה, עליך לכלול את קוד הבקרה n\ בתוך מחרוזת הפורמט, כך:

```
printf("The cost is %f for %d items\n", amount, count);
```

הצב את הקוד n\ במקום בו אתה רוצה שתתחיל שורה חדשה (לא בהכרח בסוף). ההוראה

```
printf("The cost is %f\nfor %d items\n", amount, count);
```

מציגה שתי שורות:

```
The cost is 45.580000
for 5 items
```

באפשרותך להשתמש בקודים אחרים של בקרה כדי לשלוט במספר הרווחים בשורה, לצלצל בפעמון או להציג תווים מיוחדים.

המרת טיפוסי נתונים

קיימים שני מצייני פורמט נוספים, הממירים מספר שלם למקבילה שלו בכתיב אוקטלי או הקסדצימלי:

%o	ממיר ערך לשיטה אוקטלית (השתמש באות o קטנה, לא בספרה 0).
%x	ממיר ערך לשיטה הקסדצימלית.

כדי להמיר ערך כלשהו משיטה עשרונית לפורמט אחר, השתמש במצייני הפורמט %o או %x במחרוזת הבקרה, ובמספר העשרוני ברשימת הנתונים. התוכנית הבאה, לדוגמה, מדפיסה את המקבילות של המספר העשרוני 17 בכתיב הקסדצימלי ובכתיב אוקטלי:

```
main()
{
    printf("%d is %x in hex and %o in octal/n", 17,17,17);
}
```

הפלט יהיה

```
17 is 11 in hex and 21 in octal
```

זכור שאתה זקוק למספר האוקטלי כדי להדפיס תווים וסמלים גרפיים. אם ידוע לך המספר העשרוני של התו שברצונך להדפיס, השתמש בתוכנית כמו זו כדי להמיר אותו לאוקטלי.

תווים בעלי פיצול אישיות

כפי שלמדת בפרק העוסק בפונקציה putchar()‎, ניתן להכריז על משתנים מטיפוס char גם כמשתייכים לטיפוס int. כלומר, ניתן לייחס אות ל-int ולהציג אותה באמצעות putchar()‎ או printf()‎, כדלקמן:

```
main()
{
    int a;
    a='A';
    putchar(a);
    putchar('\n');
    printf("%c",a);
}
```

כשתריץ תוכנית זו, האות A תופיע פעמיים. היא מוצגת כתו תוך שימוש ב-putchar()‎, וכתו תוך שימוש במציין %c ב-printf()‎. אולם בין אם אתה מכריז על התו כמשתייך לטיפוס char או לטיפוס int, באפשרותך להשתמש במציין %d כדי להמיר את התו לקוד ה-ASCII שלו:

```
main()
{
    char a;
    a='A';
    printf("The ASCII code of %c is %d\n",a,a);
}
```

במקרה זה, אותו משתנה מוצג באמצעות המציינים %c ו-%d. הפלט הוא:

```
The ASCII code of A is 65
```

הוא מציג את התו A תוך שימוש במציין %c ואת המספר 65 - קוד ASCII של האות A - באמצעות המציין %d.

התוכנית היתה רצה בהצלחה ומציגה בדיוק אותו פלט גם אם המשתנה a היה מוכרז כטיפוס int.

קביעת תצורת הפלט

ניתן להשתמש בפונקציה printf()‎ גם לצורך קביעת התצורה של הנתונים המוצגים. באפשרותך לשלוט בריווח השורה ובמספר התווים המוצגים בעזרת *מצייני רוחב-שדה*.

ללא מציין רוחב, מספרים מטיפוס float, לדוגמה, יוצגו עם שש ספרות אחרי הנקודה העשרונית. זו הסיבה שההוראה

```
printf("The cost is %f for %d items", amount, count);
```

תציג את הפלט

```
The cost is 45.580000 for 5 items
```

בהתאם למערכת המחשב שברשותך, ובהתאם לאופן חישוב הערך, הפלט עשוי גם להיראות כך:

```
The cost is 45.579998 for 5 items
```

אנו עושים שימוש במצייני רוחב-שדה כדי להתאים את אופן הצגת המלל והמספרים לצרכינו.

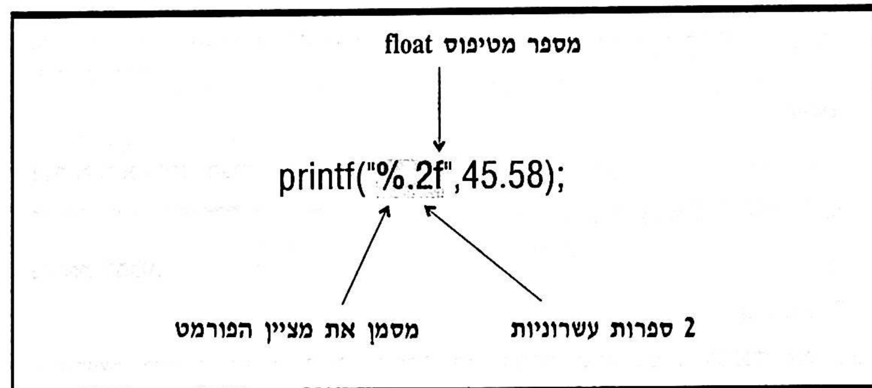
כדי לציין את מספר הספרות העשרוניות, השתמש בפורמט

```
%.nf
```

כאשר n הוא מספר הספרות העשרוניות הרצוי (ראה תרשים 4.12). לדוגמה,

```
printf("The cost is %.2f", amount);
```

תרשים 4.12
ציון מספר הספרות אחרי הנקודה העשרונית.



תציג את הסכום מטיפוס float עם שתי ספרות בלבד אחרי הנקודה העשרונית:

```
The cost is 45.58
```

באופן דומה, ההוראה

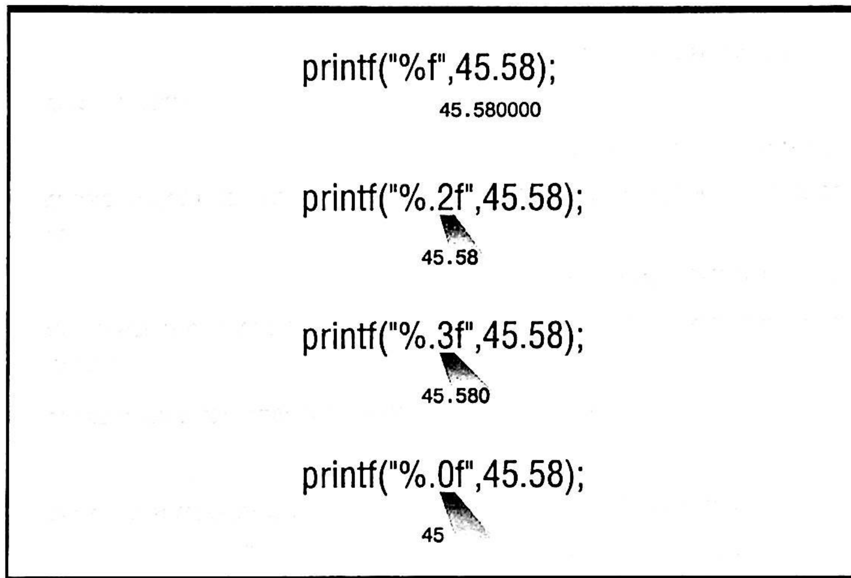
```
printf("The cost is %.3f", amount);
```

תציג את הסכום עם שלוש ספרות אחרי הנקודה:

```
The cost is 45.580
```

תרשים 4.13 מסכם את אופן השליטה במספר הספרות העשרוניות.

תרשים 4.13
שליטה במספר הספרות העשרוניות.



באפשרותך לשלוט גם על הרוחב הכללי של השדה - מספר המקומות שתופש הערך כולו - בעזרת הפורמט:

```
%N.nf
```

כאשר N הוא רוחבו הכולל של השדה. לדוגמה,

```
printf("The cost is %8.2f", amount);
```

תציג את הפלט:

```
The cost is    45.58
```

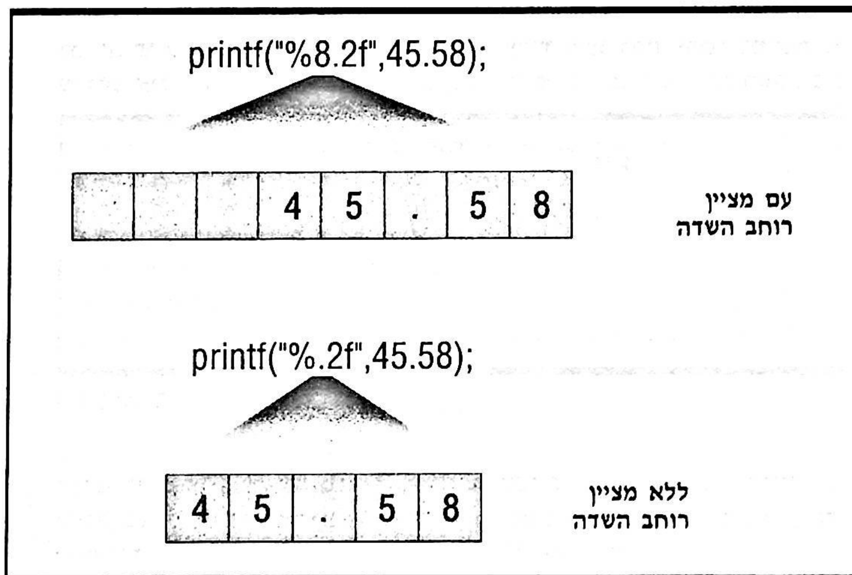
עם שלושה רווחים נוספים לפני המספר. כדי להבין זאת, דמיין לעצמך את מציין רוחב-השדה מורה לקומפיילר להדפיס את הערך בתיבה שרוחבה שמונה מקומות, כמוצג בתרשים 4.14. המספר עצמו, 45.58, תופש חמישה מקומות - מכיוון שהנקודה העשרונית נחשבת כתו לכל דבר. המקום שאינו בשימוש מופיע כרווחים לפני המספר.

גם אם תציין רוחב שדה קטן מרוחב המספר עצמו, C תציג את המספר כולו. במקום למחוק תווים מהמספר, C מתעלמת מרוחב השדה שציינת. הפקודה

```
printf("The cost is %2.2f", amount);
```

תציג את הפלט

תרשים 4.14
מציין רוחב-שדה שולט בריוות על-גבי המסך.



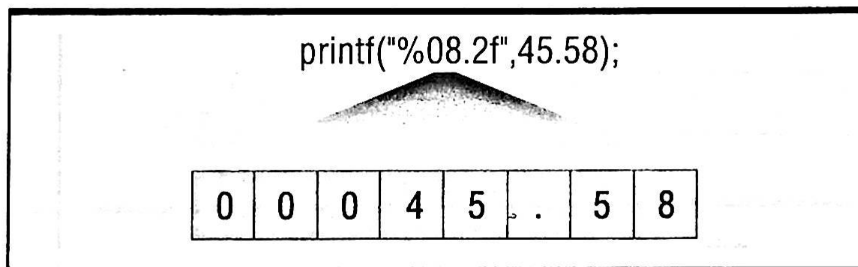
The cost is 45.58

באפשרותך גם להוסיף את המקומות הנוספים, אולם למלא אותם באפסים במקום ברווחים, כמודגם בתרשים 4.15. עליך רק לרשום 0 לפני המספר המציין את הרוחב:

```
printf("The cost is %08.2f", amount);
```

ההוראה תיצור את הפלט:

תרשים 4.15
הצגת אפסים לפני המספר.



The cost is 00045.58

כדי ליישר מספרים לשמאל - להציב אותם בצידה השמאלי של אותה תיבה דמיונית - הצב סימן מינוס אחרי ה-%, כך: %-8.2f. הרווחים הנוספים יופיעו מימין לערך:

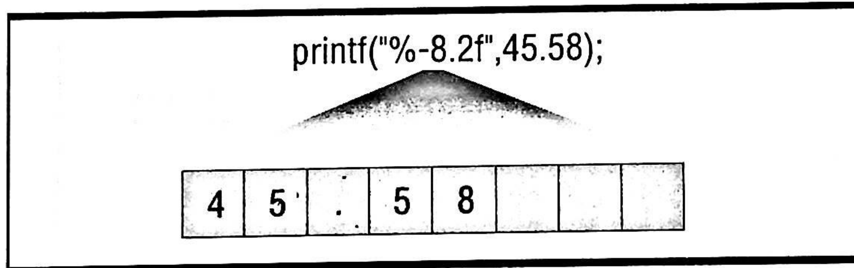
```
printf("The cost is %-8.2f in US currency", amount);
```

תציג

The cost is 45.58 in US currency

כפי שניתן לראות בתרשים 4.16, הערך 45.58 עדיין מוצג בתוך תיבה דמיונית בת ארבע עמדות, אולם הפעם הוא מופיע בצידה השמאלי של התיבה. הרווחים הנוספים מופיעים מימין לערך.

תרשים 4.16
יישור תווים לשמאל על המסך.



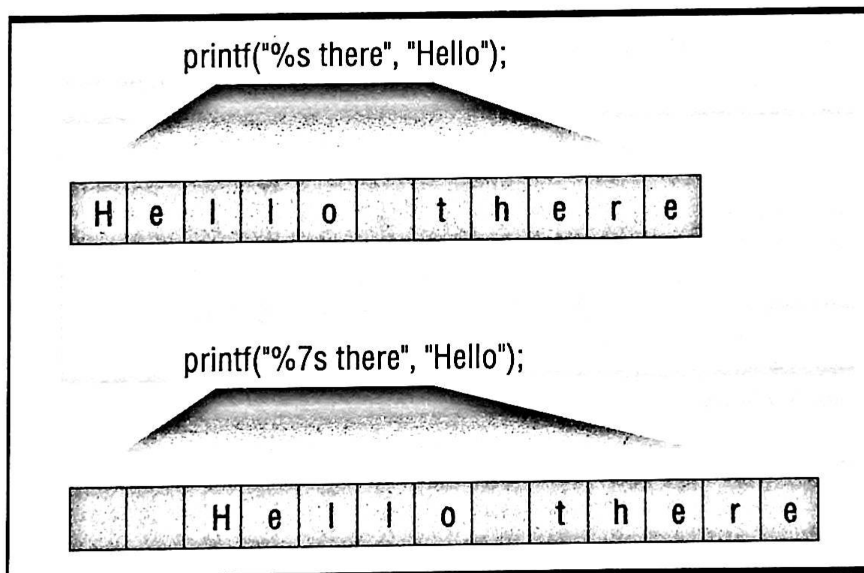
מצייני רוחב-שדה פועלים גם עם מחרוזות ונתונים תוויים. רווחים נוספים יופיעו לפני המלל, כשהם דוחקים את המלל לצידה הימני של התיבה הדמיונית. לדוגמה, אם משתנה המחרוזת ששמו message מכיל את הערך "Hello", הפקודה:

```
printf("I just called to say %7s there", message);
```

תציג את הפלט

I just called to say   Hello there

תרשים 4.17
שימוש במציין רוחב-שדה עם מחרוזות.



כפי שמודגם בתרשים 4.17, שני רווחים נוספים מוחדרים לפני ערך המחרוזת.

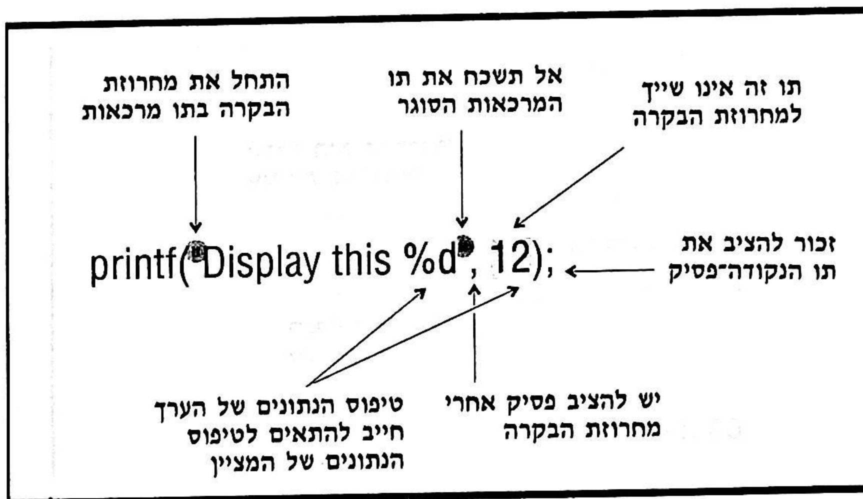
בחירת פקודת הפלט הנכונה

כאשר אתה מתכנן את הפלט, עליך לשקול ולהחליט איזו פקודת פלט או פונקצית פלט מתאימה ביותר לצרכיך.

כדי להציג מלל או תווים בלבד, השתמש בפונקציות ()puts או ()putchar. מכיוון שפונקציות אלה אינן משפיעות על התצורה הן פועלות מהר יותר, והקוד שלהן תופש פחות מקום בכונן מאשר הקוד הנוצר על ידי ()printf. בעבודה עם ()puts, בדוק תחילה אם הקומפיילר שברשותך מוסיף פקודת שורה-חדשה באופן אוטומטי. אם הקומפיילר אינו מוסיף את הפקודה, ואתה שוכח לעשות זאת, יהיה עליך להשקיע זמן נוסף בעריכת התוכנית. השמטת n\ בעבודה עם ()printf היא שגיאה נפוצה מאוד.

הפונקציה ()printf איטית יותר ותופשת מקום רב יותר בכונן. אולם זוהי הפונקציה האידיאלית כאשר ברצונך ליצור פלט של ערכים מספריים, לקבוע את הפורמט של מחרוזת, או לשלב מלל ומספרים באותה שורה. עליך רק להקליד את השורה בקפידה, ולהתאים את

תרשים 4.18
כללים לשימוש בפונקציה ()printf.



מצייני הפורמט לערכים המילוליים, לקבועים ולמשתנים ברשימת הנתונים. בתרשים 4.18 מודגמים הכללים החשובים ביותר עבור הוראות ()printf.

הפלט בשפת התכנות ++C

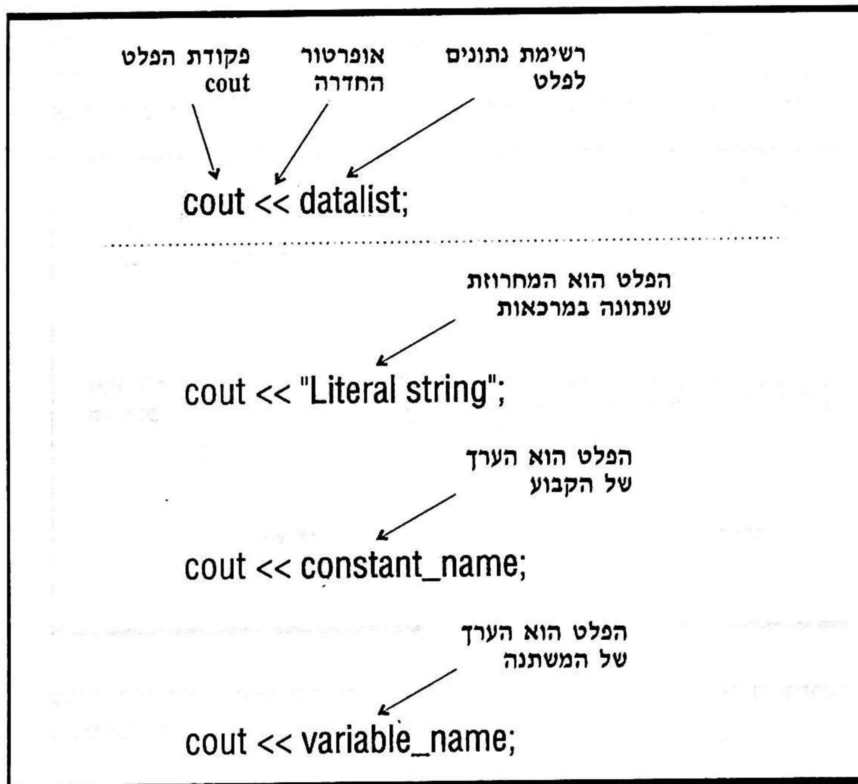
ניתן ליישם את כל טכניקות יצירת הפלט בהן דנו עד כה הן ב-C והן בשפת ++C. עם זאת, ++C כוללת דרך נוספת ליצירת פלט של כל טיפוסי הנתונים.

ההוראה cout בשפת ++C מציגה נתונים על-גבי הצג. בשילוב עם התווים <<, הנקראים אופרטור *החדרה*, cout יכולה להציג הן ערכים מילוליים והן ערכים של קבועים ומשתנים ללא צורך במצייני פורמט.

אם ברשותך קומפיילר ++C, בדוק את התיעוד הנלווה. ייתכן שעליך להשתמש בקובץ כותרת מיוחד כדי להפעיל את הפקודה cout ואת הפקודה cin, עליה תלמד בפרק הבא. בקומפיילרים מסוימים, לדוגמה, יש לציין את הקובץ ששמו iostream.h בפקודת #include בתחילת התוכנית. בפרק 2 תמצא מידע נוסף אודות הפקודה #include והשימוש בקובצי כותרת.

המבנה הבסיסי של cout מודגם בתרשים 4.19. אחרי הפקודה cout מופיעים שני התווים <<. תווים אלה מורים ל-cout להציג את המידע הבא אחריהם. המידע יכול להיות ערך מילולי הנתון במרכאות, או שמו של משתנה או קבוע.

תרשים 4.19
הפקודה cout בשפת ++C.



לדוגמה, ההוראה

```
cout << "Hi, my name is Sam. Gee, you look familiar";
```

מציגה את המחרוזת המילולית הנתונה במרכאות. ההוראות

```
int count;
count = 4509;
cout << count;
```

מציגות את המספר 4509, ערכו של המשתנה ששמו count.

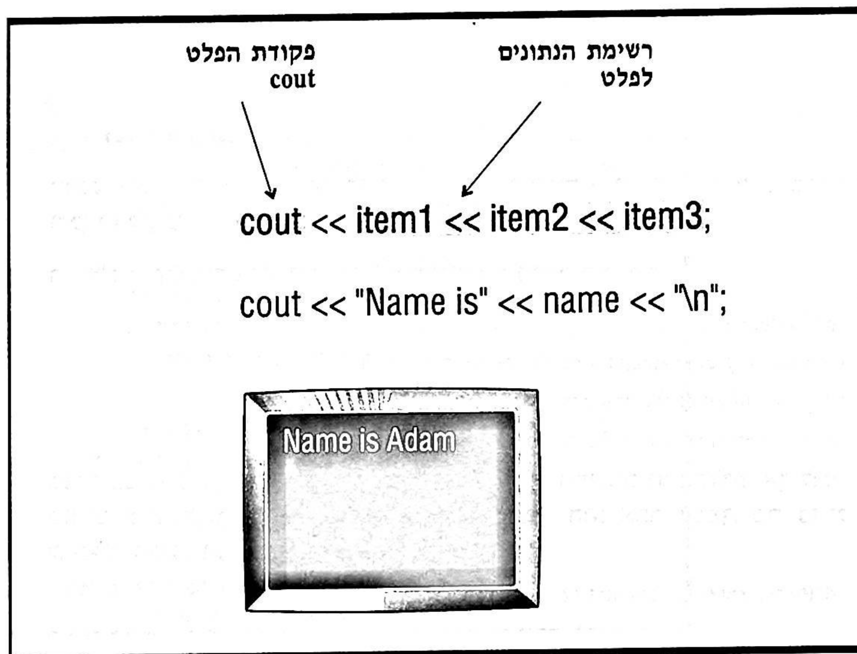
כדי להציג מספר ארגומנטים, יש להפריד ביניהם באמצעות אופרטור החדרה, כמודגם בתרשים 4.20. לדוגמה, ההוראות

```
int age;
age = 43;
cout << "You are " << age << " years old.";
```

מציגות את המלל

```
You are 43 years old.
```

תרשים 4.20
הצגת מספר ארגומנטים באמצעות cout.



הפקודה cout מציגה כל אחד מהפריטים כפי שהוא מסומן על ידי אופרטור ההחדרה, בהתאם לסדר הופעתם בהוראה.

בדומה ל-printf()‎, הפקודה cout אינה מוסיפה פקודת שורה-חדשה באופן אוטומטי לאחר הצגת הנתונים. כדי לעבור לשורה הבאה, יש להוסיף את הקוד \n במקום בו אתה רוצה שתתחיל השורה, כמודגם בתרשים 4.20.

בעבודה עם cout אין צורך במצייני פורמט, אם כי ניתן להשתמש בהם. לצורך שליטה ברוחב, בריווח ובמספר הספרות העשרוניות בעבודה עם מצייני פורמט, ניתן להשתמש בפונקציה form. אופן השימוש בפונקציה form בשפת ++C חורג מהנושאים בהם עוסק ספר זה. אם ברשותך קומפיילר ++C, בדוק את התיעוד הנלווה כדי ללמוד נושא זה ביתר פירוט. כמו כן, עיין בתיעוד כדי לדעת אם עליך לכלול קובץ כותרת מיוחד כדי להשתמש ב-cout.

העמסת-יתר (Overloading) בשפת ++C

תכונה מיוחדת של ++C הנקראת *העמסת-יתר* עושה את מצייני הפורמט למיותרים. בניגוד ל-printf()‎, הדורשת ציון מפורש של פורמט הנתונים המועבר לפונקציה, cout יכולה לקבוע זאת בעצמה, בהתבסס על טיפוס הנתונים שהיא מקבלת. תהליך זה נקרא העמסת-יתר.

עיצוב התוכנית

הפלט - כל מה שאתה משגר למסך, למדפסת וכדומה - מהווה חלק חיוני בכל תוכנית, ולכן יש לתכנן כל פלט בקפידה.

התחל את התוכנית בהוראות פלט המסבירות את מטרת התוכנית:

```
puts(" Welcome to the Mortgage Calculator\n");
puts("This program calculates mortgage payments.\n");
puts("You will enter the amount of the mortgage loan,\n");
puts("the interest rate, and the number of years of the loan.\n");
```

בפרק הבא תלמד אודות הקלט - הזנת נתונים לתוך התוכנית מהמקלדת. אך לפני בקשת נתונים עליך לומר למשתמש, אפילו אם המשתמש הוא אתה עצמך, מה בדיוק עליו להקליד. השתמש בשורת פלט עבור כל פריט קלט:

```
puts("Please enter the amount of the mortgage loan:");
```

כאשר התוצאות מוכנות להצגה, עשה את התצוגה נוחה לקריאה וברורה:

```
printf("Monthly principle payment:  %7.2f\n", princ);
printf("Interest payment:           %7.2f\n", interest);
printf("Total monthly payment:      %7.2f\n", total);
```

הרווחים הנוספים בין תווי הנקודתיים לבין מצייני הפורמט, היישור של מצייני הפורמט והמציינים עצמם, כל אלה גורמים ליישור הערכים המספריים בתצוגה, כך:

```
Monthly principle payment:    256.25
Interest payment:             92.12
Total monthly payment:        34.37
```

צורת תצוגה זו עדיפה ללא ספק על ההוראה:

```
printf("%f %f %f", princ, inter, total);
```

המציגה את הפלט כך:

```
256.25 92.12   34.37
```

את צורת המחשבה הזו יש ליישם לכל פן של התוכנית. הדבר גוזל מעט יותר זמן, אולם התוצאות מצדיקות זאת. התוכנית תיראה מקצועית יותר, ותשרת טוב יותר את המשתמשים.

טבלה 4.2 מסכמת את שיטות יצירת הפלט שנדונו בפרק זה.

טבלה 4.2
סיכום פקודות פלט

פונקציה או מלת מפתח	טיפוס נתונים	הערות
puts()	טיפוס מחרוזות בלבד	שפת C ושפת ++C. משמשת ליצירת פלט של מחרוזות בלבד. ניתן להציג ערכים מילוליים הנתונים במרכאות, משתנים וקבועים. עשויה להעביר את הסמן לשורה הבאה באופן אוטומטי, אחרי הצגת הנתונים (אולם יש לבדוק את התיעוד הנלווה לקומפיילר).
putchar()	תווים בודדים	שפת C ושפת ++C. משמשת ליצירת פלט של תו המוכרז כשייך לטיפוס הנתונים int או לטיפוס char. אינה מעבירה את הסמן לשורה הבאה באופן אוטומטי אחרי הצגת הנתונים (אולם יש לבדוק את התיעוד הנלווה לקומפיילר). השתמש בפונקציה זו להצגת תו מילולי הנתון בין גרשיים, קבוע או משתנה מטיפוס char, קוד בקרה או תו מיוחד.

טבלה 4.2
סיכום פקודות פלט *(המשך)*

פונקציה או מלת מפתח	טיפוס נתונים	הערות
printf()	כל טיפוסי הנתונים	שפת C ושפת ++C. יש לכלול מציין פורמט עבור כל פריט נתונים שברצונך להציג. ניתן להשתמש בפונקציה זו ליצירת פלט של מספר ארגומנטים בהוראה אחת. אינה מעבירה את הסמן לשורה הבאה באופן אוטומטי - יש להשתמש לשם כך בקוד n\.
cout	כל טיפוסי הנתונים	משמשת בשפת ++C בלבד. יש להפריד בין הארגומנטים באמצעות התווים <<. אינה מעבירה את הסמן לשורה הבאה באופן אוטומטי. אין הכרח להשתמש במצייני פורמט.

שאלות

1. מהו פלט?
2. באיזה שלושה סוגים של ארגומנטים ניתן להשתמש בפרמטר של ()puts?
3. באיזה שלושה סוגים של ארגומנטים ניתן להשתמש בפרמטר של ()putchar?
4. מהם קודים של בקרה?
5. מה ההבדל בין קוד הבקרה n\ לבין קוד הבקרה r\?
6. כיצד מציגים את תו המרכאות על המסך?
7. מהם שני חלקיה של הוראת ()printf?
8. מהם היתרונות של ()printf לעומת ()puts?
9. מהו מציין פורמט?
10. כיצד מציגים את ערכו של משתנה מספרי?

תרגילים

1. כתוב הוראות ()puts להדפסת שמך וכתובתך.

2. כתוב הוראות ()printf להדפסת שמך וכתובתך.

3. כתוב הוראת ()puts שממרכזת את המלה Title על-גבי המסך. רוחב המסך הוא 80 תווים.

4. כתוב הוראת ()printf המציגה את המלה Page בצידו הימני של המסך.

5. כתוב הוראת ()printf אחת שיוצרת פלט של ערכי המשתנים הבאים:

```
float length, width, height, volume;
```

6. כתוב הוראת ()printf המציגה את ערכי המשתנים הבאים עבור תוכנית המציגה את שמו וגילו של אדם כלשהו:

```
char name[12];
int age
```

7. תוכנית מסוימת כוללת את המשתנים הבאים:

```
char item[] = "Floppy disk";
float cost = 3.55;
float markup = 0.75;
```

כתוב הוראות ()printf המציגות את הפלט הבא:

```
Item Name: Floppy disk
Item Cost:      3.55
Markup :        0.75
```

שים לב ליישור הערכים המוצגים.

8. תוכנית מסוימת כוללת את המשתנה הבא:

```
int count = 30;
```

כתוב את הוראות הפלט המשמיעות צלצול פעמון, ולאחר מכן מציגות את ההודעה הבאה, תוך שימוש בערכו של המשתנה count כדי להציג את המספר בשורה האחרונה:

```
Warning! Warning! Warning! Warning!
An intruder has been detected.
You have 30 seconds to leave the premises.
```



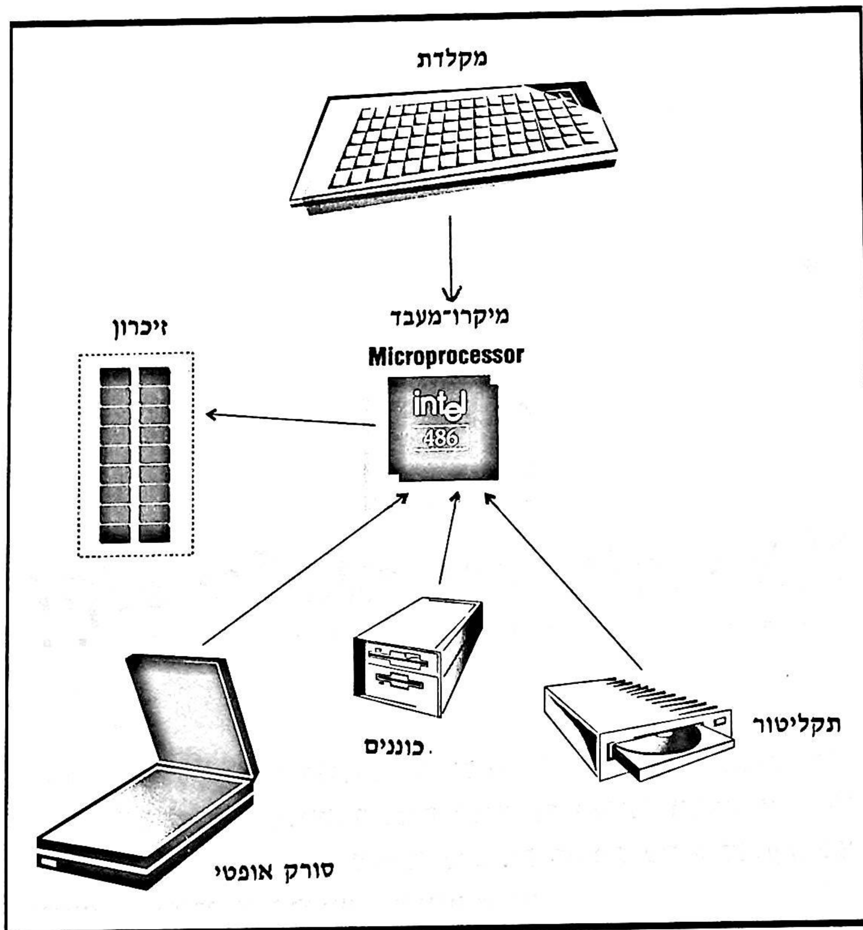
5

הקלט בשפות התכנות ++C/C



לט הוא תהליך הזנת הנתונים עבור התוכנית למחשב. הנתונים מוזנים לתוך משתנה. כלומר, הנתונים שהמשתמש מקליד, בתגובה לבקשת קלט שמציגה התוכנית, הופכים לערך של משתנה המאוחסן בזיכרון. לאחר מכן, התוכנית עושה שימוש במשתנה בפעולות השונות שהיא מבצעת. כפי שמודגם בתרשים 5.1, הקלט יכול להגיע ממקורות שונים.

תרשים 5.1
קלט מכל מקור שהוא מאוחסן במחשב כמשתנה.

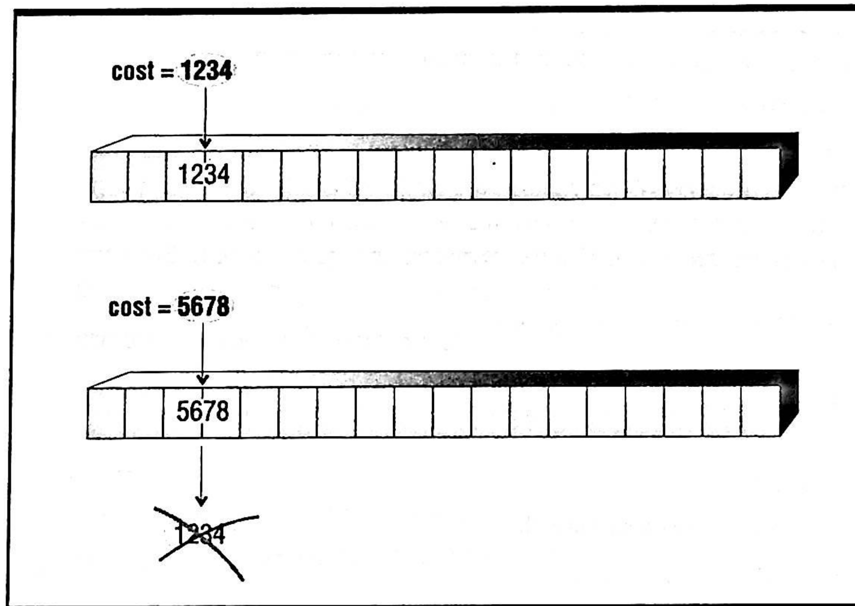



הערה
כפי שתראה בהמשך, המלה **קלט** משמשת בשתי הוראות שונות. כאשר משתמשים בה כפועל, הכוונה היא לתהליך הזנת הנתונים למחשב. כשמשתמשים במלה קלט כשם עצם, הכוונה היא לנתונים המוזנים עצמם.

התוכנית חייבת לקבל את הקלט המתאים. זכור את הפתגם הידוע: *נכנס זבל - יצא זבל.* אם הקלט המוזן לתוכנית שגוי, הפלט יהיה אף הוא שגוי. מידת הדיוק של הפלט לעולם לא תעלה על זו של הקלט.

בפרק זה תלמד כיצד לקבל נתונים מהמקלדת. בפרק 12 נעסוק באופן בו קוראת התוכנית נתונים מקובץ.

לא ניתן להזין נתונים לתוך קבוע, אלא למשתנה בלבד. הקבוע יכיל תמיד את הערך ההתחלתי שהקצית לו. כאשר אתה מזין ערך כלשהו למשתנה, אתה מציב את הערך בעמדת הזיכרון המתייחסת לאותו משתנה. אם המשתנה מכיל כבר ערך כלשהו, הערך החדש שאתה מזין (או מקצה למשתנה באמצעות סימן השווה) תופש את מקומו (תרשים 5.2). הערך הקודם נעלם לבלי שוב.



תרשים 5.2
תוכנו של המשתנה מוחלף כאשר מזינים ערך חדש.

הערות ++C

++C תומכת בכל פקודות הקלט של שפת C.

הפונקציה ()gets

הפונקציה ()gets מזינה מחרוזת לתוך משתנה. הפרמטר הוא שמו של המשתנה. לדוגמה, עיין בתוכנית הבאה:

```
main()
{
    char name[15];
    gets(name);
    puts(name);
}
```

הפקודה ()gets, במקרה זה, תקצה את עד 14 התווים הראשונים שיוקלדו במקלדת למשתנה מחרוזת ששמו name. זכור ש-C מקצה עמודות זיכרון בהתאם למספר המפורט בהכרזה. מכיוון שעמדת זיכרון אחת שמורה עבור מסיים האפס, באפשרותך להקליד מספר

תווים הקטן באחד מהמספר שצוין בהכרזה. אם ברצונך להזין שם בן 15 תווים, לדוגמה, עליך להכריז על שם המשתנה כבעל 16 תווים לכל היותר:

```
char name[16]
```

כאשר מתבצעת הוראת ()gets, התוכנית נעצרת. היא ממתינה שהמשתמש יקליד דבר-מה. הנתונים המוקלדים הופכים בפועל לערך של משתנה רק לאחר הקשת Enter. עם הקשת Enter, המחרוזת שהקליד המשתמש מוקצית למשתנה, והסמן עובר לשורה הבאה במסך. התו Enter עצמו אינו הופך חלק מהמחרוזת, אולם C מוסיפה את מסיים האפס 0\ כדי להשלים את המחרוזת.

הבה נתבונן בדוגמה נוספת, מפורטת יותר:

```
main()
{
    char name[25];
    printf("Please enter your name: ");
    gets(name);
    printf("Confirming, your name is: %s", name);
}
```



הערה

כאשר אתה מקליד תווים בתגובה להוראת ()gets, התווים המוקלדים מופיעים אמנם על-גבי המסך, אולם אין זה כתוצאה מפקודת פלט כלשהי. למעשה, התווים אינם מוזנים למחשב בפועל עד להקשת Enter.

כשתריץ את התוכנית, תופיע שורת ההנחיה הבאה:

```
Please enter your name:
```

מכיוון שהשתמשנו בפקודה ()printf ללא הקוד n\, הסמן נשאר בסופה של ההנחיה. הסמן יעמוד במרחק של תו אחד מתו הנקודתיים, מכיוון שהכנסנו רווח בין הנקודתיים לבין תו המרכאות הסוגר את הפרמטר של ()printf. אם תמשיך לשבת ולהתבונן במסך, לא יקרה דבר. התוכנית ממתינה לקלט כלשהו - במקרה זה, שתקליד את שמך.



טיפ

התרגל להשתמש בביטויי ()printf או ()puts המורים למשתמש מה עליו להקליד לפני כל פקודת קלט. כפי שתראה בהמשך, הצגת הנחיות ברורות היא בעלת חשיבות עליונה בפקודות קלט אחרות.

תוך כדי הקלדת שמך, התווים מופיעים על-גבי המסך. אם שגית בהקלדה וטרם הקשת Enter, באפשרותך להקיש על מקש backspace כדי למחוק את התווים השגויים ולהקלידם מחדש. במערכות מחשב מסוימות ניתן להקיש Esc כדי למחוק בבת אחת את כל התווים שהוקלדו, ולהתחיל מחדש.

הקשת Enter מציבה את התווים שהוקלדת במשתנה name, ומחדירה את המסיים אפס בסופה של המחרוזת. לאחר מכן מתבצעת פקודת ()printf השנייה. אם שמך הוא Alvin Aardvark, יוצג על המסך המלל הבא:

```
Please enter your name: Alvin Aardvark
Confirming, your name is: Alvin Aardvark
```

זכור, שמך מופיע בסופה של שורת ההנחיה הראשונה רק בגלל שהתווים שהקלדת כקלט מוצגים על-גבי המסך. תווים אלה לא נקלטו בתוכנית עד שהקשת Enter. בשורת ההנחיה השנייה התווים מוצבים כמשתנה על ידי הפקודה ()printf.

הפונקציה ()gets משמשת דרך מצוינת להזנת מחרוזות לתוכנית.

הפונקציה ()getchar

הפונקציה ()getchar מזינה תו יחיד מהמקלדת. מרבית הקומפיילרים מאפשרים להזין את התו הן כתו המשתייך לטיפוס char והן כתו המשתייך לטיפוס int, בגלל האופן בו מגדיר תקן K&R משתני תווים. (בפרק 4 תמצא מידע נוסף בנושא פיצול האישיות של תווים).

כדי להזין תו, השתמש באחד הפורמטים הבאים:

```
char letter;                int letter;
letter = getchar();         letter = getchar();
```

שים לב שהקריאה לפונקציה ()getchar שונה מהקריאה לפונקציות האחרות שהכרנו עד עתה. במקום להופיע בתחילת השורה, היא מיוחסת למשתנה באמצעות סימן השווה (=). פורמט זה פירושו: "התוצאות המתקבלות מביצוע הפונקציה ()getchar מוצבות כערך במשתנה letter". למעשה, הקריאה לפונקציה ()getchar מטופלת כאילו היתה ערך בעצמה (תרשים 5.3). כאשר השורה מתבצעת, נקראת הפונקציה ()getchar, מוזן תו יחיד, והתוצאה מוצבת במשתנה. *פונקציה זו אינה מקבלת ארגומנט*. כלומר, לא מועבר ערך כלשהו לפונקציה בסוגריים.



הערה

קומפיילרים מסוימים של C ושל ++C עושים שימוש בפונקציה ()getch כדי להזין תו מבלי להזדקק להקשת Enter. בקומפיילרים מסוג זה, כאשר התוכנית קוראת את התו באמצעות הפונקציה ()getchar, המשתמש חייב לעתים להקיש Enter אחרי הקלדת התו. עיין בתיעוד הנלווה לקומפיילר שברשותך.

כאשר המשתמש מקליד תו כלשהו, הפונקציה ()getchar מציגה את התו על המסך. אין צורך להקיש Enter, מכיוון ש-()getchar מקבלת תו אחד בלבד - התוכנית ממשיכה לרוץ ברגע שהוקלד תו אחד. התו מוצב במשתנה מייד עם הקלדתו.

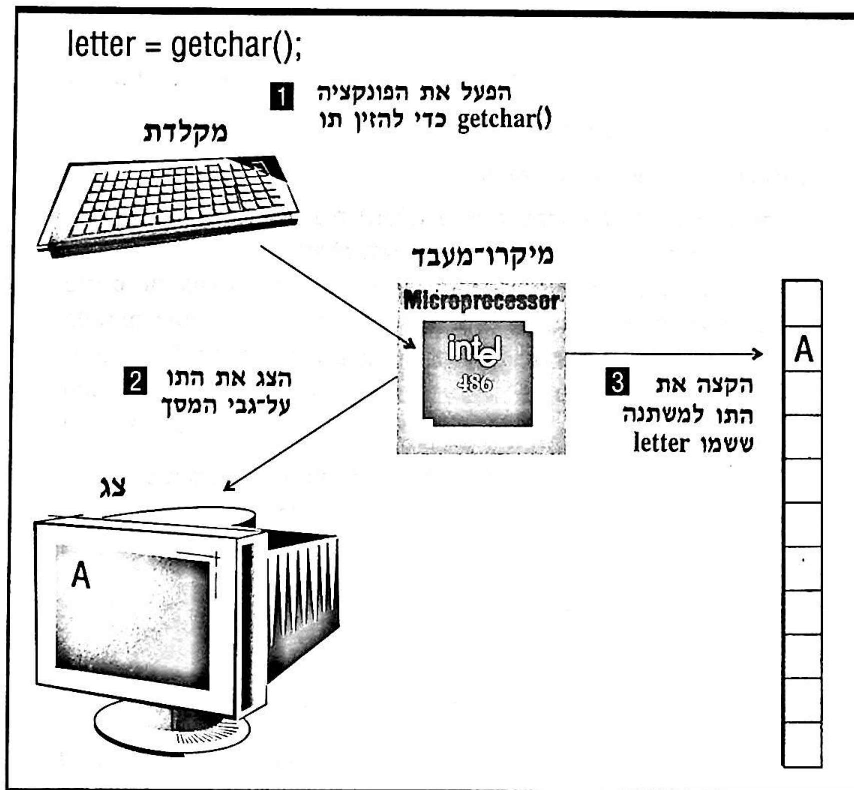
מתי מתעורר הצורך להזין תו יחיד? נתקלת בוודאי בתוכניות הדורשות תשובה של Yes או No, או שמורות לך לבחור פריט מסוים מתוך תפריט. אלו הם יישומים המתאימים במיוחד לפונקציה ()getchar, מכיוון שהיא אינה ממתינה להקשת Enter. די בהקשת התו כדי להשלים את הקלט ולהמשיך בהרצת התוכנית.

גם כאשר התו המוזן באמצעות ()getchar משתייך לטיפוס הנתונים int, ניתן להציגו בעזרת הפונקציה ()putchar, או על ידי שימוש במציין c% בהוראת ()printf:

```
/*getchar.c*/
main()
{
      int initial;
      puts("Please enter your middle initial.");
```

```
    initial = getchar();
    putchar('\n');
    putchar(initial);
    putchar('\n');
    printf("%c",initial);
}
```

תרשים 5.3
הפונקציה ()getchar.



תוכנית זו מזינה תו למשתנה מטיפוס int ששמו initial, ומציגה את התו באמצעות הפונקציות ()putchar ו-()printf. אם תקיש את האות J בתשובה להנחיה, תוצג האות J על המסך שלוש פעמים בשורות נפרדות - פעם אחת תוך כדי ההקלדה, ופעמיים נוספות כתוצאה מהוראות הפלט. במרבית מערכות המחשב לא מתבצעת פקודת שורה חדשה באופן אוטומטי אחרי הקלדת התו.

באפשרותך גם ליצור פלט של קוד ה-ASCII של התו, בעזרת המציין d%, כך:

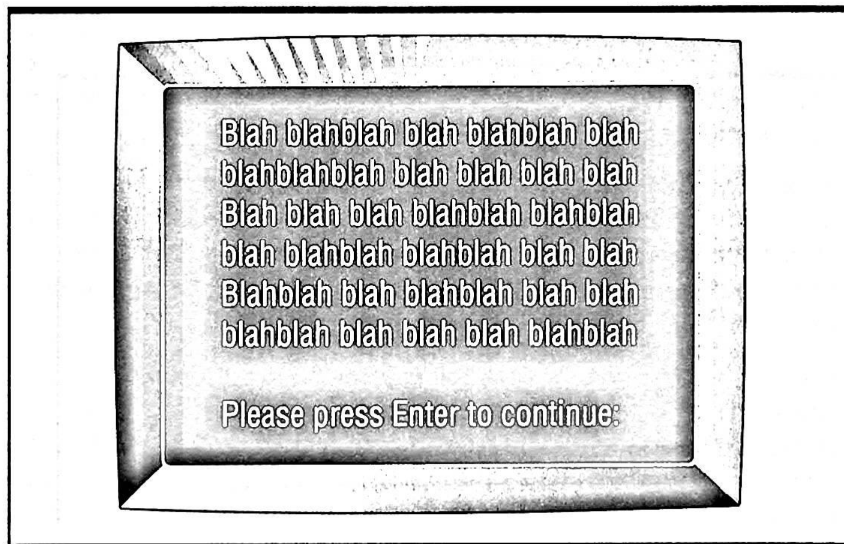
```
/*ascii.c*/
main()
{
      int letter;
      letter = getchar();
      printf("The ASCII code of %c is %d\n",letter,letter:
}
```

בדוגמה זו, המשתנה letter מוזן באמצעות הוראת ()getchar. הפקודה ()printf מציגה את המשתנה בשני פורמטים. המציין c% מציג את התו כפי שהוקלד במקלדת. המציין d% מציג את המשתנה כמספר שלם, קוד ASCII המקביל לאותה אות.

גם אם תכריז על המשתנה כמשתייך לטיפוס int ותזין ספרה כלשהי, לא תוכל להשתמש בספרה בפעולה מתמטית. כל עוד השתמשת בפונקציה ()getchar כדי להזין את התו, ההתייחסות אליו היא כאל תו ולא כאל ערך מספרי.

השהיית תוכנית ("Press Enter to Continue")

ניתן להשתמש בפונקציה ()getchar כדי להשהות את התוכנית. לדוגמה, מסך המחשב יכול



תרשים 5.4 שימוש בפונקציה ()getchar להשהיית התצוגה.

להציג מספר מוגבל של שורות בעת ובעונה אחת - בדרך כלל 25 שורות. אם השתמשת בתוכנית בסדרה של הוראות ()puts המציגות על המסך מספר שורות גדול יותר ממספר השורות שניתן להציג בו זמנית, השורות הראשונות שהוצגו ייעלמו מחלקו העליון של המסך. במקרה כזה, ייתכן שלא יהיה די זמן למשתמש לקרוא את המידע הכלול בשורות שנעלמו.

כדי להתגבר על הבעיה, כתוב פקודות ()puts - או פקודות פלט אחרות - במספר שימלא רק חלק מהמסך, והוסף את ההוראות

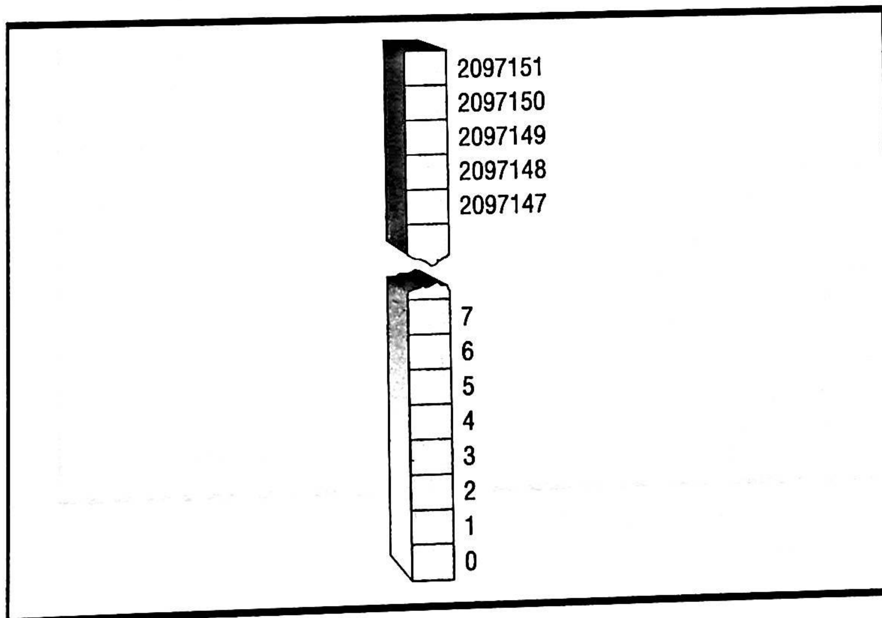
```
printf("Press Enter to continue");
getchar();
```

ההוראה ()getchar, המשמשת כאן לבדה וללא פרמטר, גורמת להשהיית התוכנית עד להקשה על מקש כלשהו (תרשים 5.4). למעשה, המקש אינו חייב להיות Enter, אולם הקשה על מקש אחר תציג את התו על המסך, דבר העלול להטעות את המשתמש. כשמשתמשים ב-()getchar באופן כזה, לא מאוחסן ערך כלשהו במשתנה. אם תקיש את האות Y למשל כדי להמשיך בהרצת התוכנית, תוצג האות Y על המסך, אולם היא לא תוצב במשתנה כלשהו.

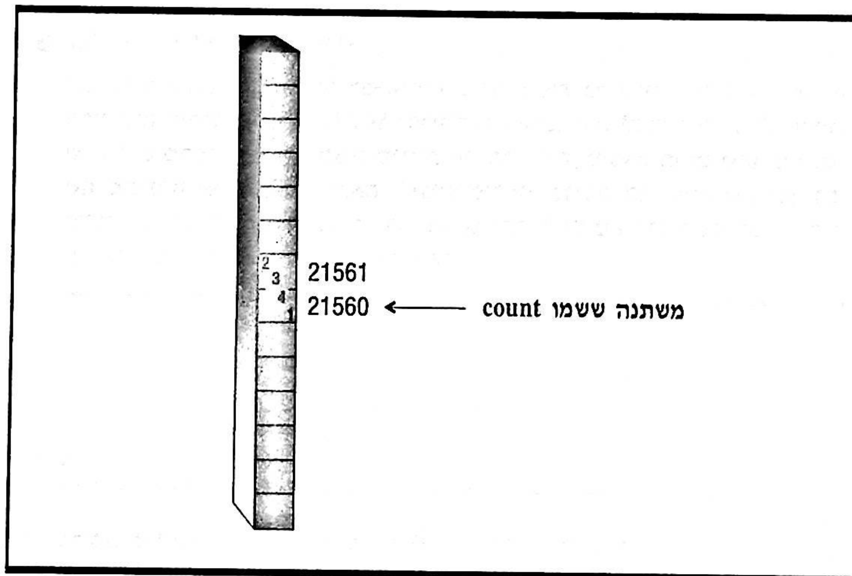
אופרטור הכתובת &

כידוע לך, כל משתנה בתוכנית תופס מקום בזיכרון המחשב. (כמות הזיכרון שתופס המשתנה תלויה בטיפוס הנתונים אליו הוא משתייך. עיין בפרק 3 כדי לרענן את זיכרונך). כל עמדה בזיכרון ממוספרת, החל מ-0 ועד לעמדת הזיכרון האחרונה במערכת המחשב

תרשים 5.5
כתובות זיכרון.



תרשים 5.6
משתנה וכתובתו.



שלך. אם המחשב שלך מכיל 2 מגהבייט של זיכרון, העמדות ממוספרות מ-0 עד 2,097,151 – כמודגם בתרשים 5.5. מספרים אלה נקראים *כתובות זיכרון*.

כאשר אתה מכריז על משתנה, C מקצה די עמדות זיכרון כדי לאחסן את הערך. אם הכרזת על משתנה מטיפוס int ששמו count, לדוגמה, C מקצה שתי עמדות זיכרון כדי לאחסן את הערך שיוצב ב-count. העמדה הראשונה מבין עמדות הזיכרון האלה היא כתובתו של המשתנה.



הערה
כתובותיהם האמיתיות של משתנים תלויות במחשב, ועשויות אף להשתנות בכל פעם שאתה מריץ את התוכנית. השתמשנו כאן בכתובת 21560 כדוגמה בלבד.

בתרשים 5.6 ערכו של המשתנה count הוא 2341. ערך זה מאוחסן בעמדות הזיכרון 21560 ו-21561. כתובתו של המשתנה count, להבדיל מערכו, היא לפיכך 21560. די לקבוע את העמדה הראשונה בתור כתובת, מכיוון ש-C יודעת כמה עמדות זיכרון תופס כל טיפוס של משתנה. אם count הוא משתנה מטיפוס int התופס שתי עמדות זיכרון, והוא מתחיל בעמדה מספר 21560, הרי שהוא חייב להסתיים בעמדה מספר 21561.

כאשר אתה רוצה להתייחס לערך של משתנה כלשהו, כדי להציגו בפקודת פלט, למשל, השתמש בשמו של המשתנה. כדי להתייחס לכתובתו של המשתנה, יש להציב את התו & לפני שמו של המשתנה, כך: &count. התו & נקרא *אופרטור הכתובת*. הוא מורה ל-C שאתה מעוניין בכתובת בה מאוחסן המשתנה, ולא בערך המאוחסן בעמדה זו. לפיכך, בעוד שערכו של המשתנה count הוא 2341, כתובתו של &count היא 21560.

אין להשתמש באופרטור הכתובות עם משתני מחרוזת. מחרוזות מאוחסנות במשתנים מסוג מיוחד, הנקרא מערך, עליו נלמד בפרק 10.

הפונקציה ()scanf

הפונקציה ()scanf היא דרך רב-שימושית להזין את כל סוגי המידע למחשב. שמה של הפונקציה מורכב מהמלים SCAN (סריקה) ו-Formatted (תצורה קבועה), כלומר, סריקה של תווים מהמקלדת. הפונקציה סורקת את המקלדת, מוצאת תווים שיש להזינם למחשב, ואז מתרגמת את הקלט בהתאם למצייני פורמט. בדומה לפונקציה ()printf, גם ()scanf יכולה לטפל במספר ארגומנטים, ולכן באפשרותך להזין משתנים מספריים, משתני מחרוזת ומשתנים מטיפוס char, בעת ובעונה אחת.

הערות ++C

++C תומכת בכל פקודות הקלט של C, ובנוסף - כוללת גם פונקצית קלט רב-שימושית הנקראת cin. עיין בפסקה "הקלט בשפת התכנות ++C" בהמשך הפרק.

בדומה ל-()printf, גם הפרמטר של ()scanf מורכב משני חלקים, מחרוזת בקרה ורשימת נתונים. (ראה תרשים 5.7) מחרוזת הבקרה כוללת מצייני פורמט המורים כיצד יש לתרגם את נתוני הקלט. בפונקציה ()scanf נקראים המציינים *תווי המרה*(conversion characters). רשימת הנתונים מצביעה על המשתנים שיאחסנו את הערכים המוזנים לתוכנית.

תרשים 5.7
הפרמטר של ()scanf מורכב משני חלקים.

רשימת נתונים **מחרוזת בקרה**

```
scanf("          ",          );
```

()scanf עושה שימוש במצייני הפורמט שמשמשים את הפונקציה ()printf:

%d	מציג מספר שלם
%u	מציג מספר שלם בלתי-מסומן
%f	מציג מספר עשרוני, מטיפוס float
%e	מציג מספר בכתיב מדעי
%g	מציג מספר עשרוני בכתיב עשרוני או בכתיב מדעי, הקצר מביניהם
%c	מציג תו מטיפוס char
%s	מציג מחרוזת
%o	מציג מספר אוקטלי (בסיס 8)

%x מציג מספר הקסדצימלי (בסיס 16)

כאשר מזינים נתונים מספריים או נתונים מטיפוס char, חובה לציין ברשימת הנתונים גם את כתובת המשתנה ולא רק את שמו:

```
main()
{
        float amount;
        scanf("%f", &amount);
}
```

ההוראה ()scanf תזין ערך מטיפוס float לעמדת הזיכרון בא מאוחסן המשתנה amount. כאשר מציבים ערך בכתובת של משתנה, הערך מיוחס למשתנה באופן אוטומטי.

כאשר פונקצית ()scanf מתבצעת, התוכנית ממתינה שתקליד נתונים. התווים שתקיש במקלדת יופיעו על-גבי המסך, עד שתקיש Enter.

אופן הזנת הנתונים על ידי הפונקציה ()scanf שונה לחלוטין מאופן הזנת הנתונים באמצעות הפונקציות ()gets ו-()getchar. כדי להבין כיצד פועלת ()scanf, עלינו לבחון את ההבדלים האלה ביתר פירוט.

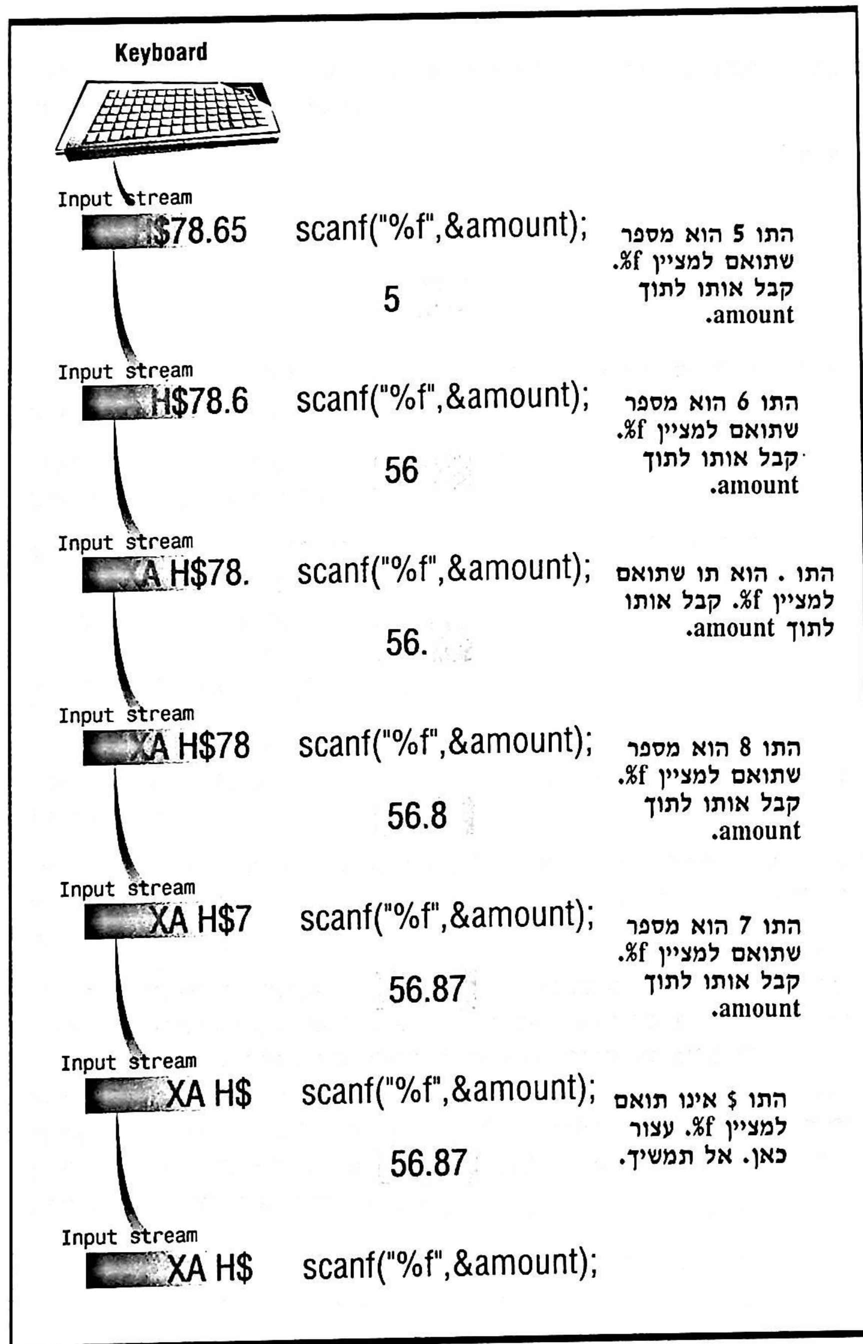
שטף קלט (Stream)

בהזנת נתונים לפקודה ()gets, כל התווים המוקלדים עד להקשת Enter מוצבים במשתנה. כאשר מזינים תו באמצעות הפונקציה ()getchar, התו המוקלד מיוחס למשתנה באופן אוטומטי.

הפונקציה ()scanf פועלת באופן שונה. במקום לקחת את הנתונים המוקלדים ולייחס אותם למשתנה, ()scanf משתמשת במצייני הפורמט כדי לתרגם את האופן בו יש להשתמש בתווים המוקלדים.

()scanf לוקחת את הנתונים מתוך *שטף קלט*. שטף הוא סדרה של תווים המוזנים ממקור כלשהו. במקרה של ()scanf, המקור הוא המקלדת. כאשר אתה מקיש Enter אחרי הקלדת הנתונים, כל מה שהקלדת מועבר לפונקציה ()scanf כסדרה של תווים חסרי משמעות, בסדר בו הקלדת אותם. על ()scanf לקבוע איזה תווים מתאימים לטיפוס הנתונים עליו מצביע מציין הפורמט, ומאיזה תווים עליה להתעלם. מצייני הפורמט נקראים תווי המרה מכיוון שהם ממירים את התווים הגולמיים בשטף הקלט לנתונים, בהתאם לטיפוסי הנתונים שציינת. תרשים 5.8 מדגים את התהליך.

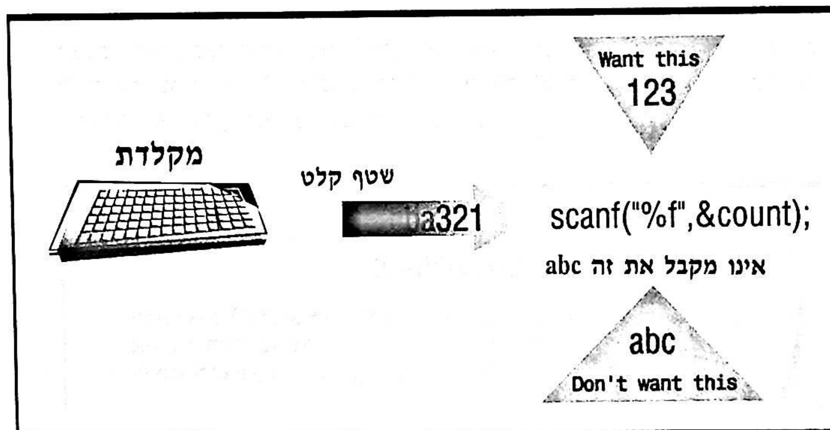
תרשים 5.8
scanf() קוראת את שטף הקלט כדי לקבוע איזה נתונים לקבל ומאיזה נתונים להתעלם.



scanf() מתעלמת באופן אוטומטי מכל תווי הרווח - רווחים, תווי דילוג או פקודת שורה-חדשה - אלא אם כן פריט הנתונים הבא משתייך לטיפוס char. התבונן בתוכנית הבאה:

```
main()
{
    int count;
    puts("Please enter a number: ");
    scanf("%d", &count);
    printf("The number is %d",count);
}
```

באפשרותך להקיש על מקש הרווח כמה פעמים שתרצה לפני הקלדת המספר. C מתעלמת מהרווחים ומחפשת את התו הראשון שאינו רווח. לאחר מכן היא מנסה להתאים את התווים לפורמט עליו מצביע המציין. אם התווים מתאימים לפורמט - בדוגמה שלנו, אם אלו הם מספרים - scanf() מציבה את המספרים במשתנה. הצבת המספרים במשתנה נפסקת ברגע שהפונקציה נתקלת בתו שאינו מתאים לטיפוס הנתונים, ובדוגמה שלנו - תו שאינו מספר. לפיכך, אם הקלדת 123abc, הערך 123 יוצב במשתנה, והפונקציה תתעלם מהאותיות abc, כמודגם בתרשים 5.9. ההצבה במשתנה נעצרת גם ברגע שהפונקציה נתקלת בתו רווח. אם הקלדת 3 12, הערך 12 יוצב במשתנה והפונקציה תתעלם מהמספר 3.



תרשים 5.9
scanf() מפסיקה לקבל נתונים ברגע שהיא נתקלת בתו שאינו מספר.

התו הראשון בשטף הקלט שאינו תו רווח חייב להתאים למציין הפורמט. אם תקליד ABC123, הפונקציה תתעלם מהעיול (entry) כולו, ולא תוכל לדעת איזה ערך מאוחסן במשתנה.

התווים הנחשבים "מתאימים" תלויים במציין. אם המציין הוא %d, לדוגמה, רק תווי הספרות ותו הקו המפריד (סימן המינוס) מותרים. אם המציין הוא %x, מותרים התווים 0123456789ABCDEF, מכיוון שאלו הם התווים המשמשים במספרים הקסדצימלים. אם

המציין הוא c%, כל תו ייחשב "מתאים", כולל תו הרווח. מכיוון שטיפוס הנתונים char יכול להכיל כל תו, ()scanf אינה מתעלמת מתווי רווח בשטף הקלט. אם ההוראה היא

```
char letter;
scanf("%c", &letter);
```

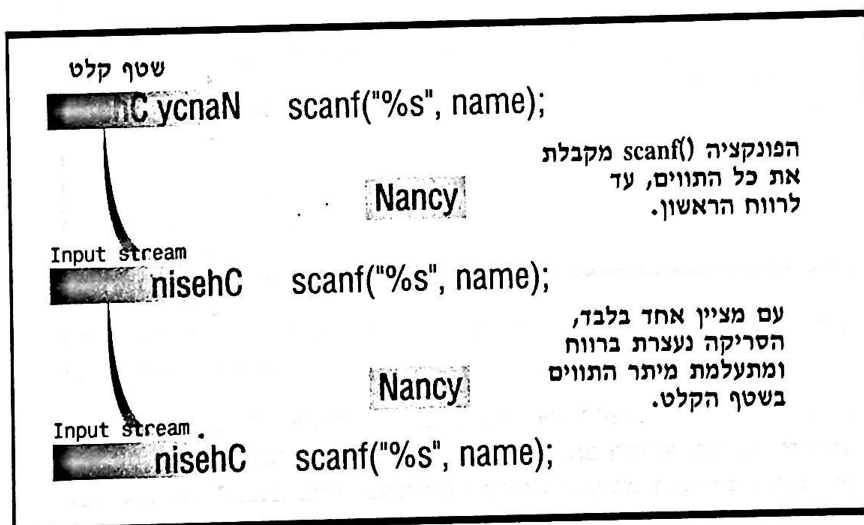
ואתה מקיש על מקש הרווח לפני הקשת התו, ()scanf תציב את תו הרווח במשתנה ותתעלם מהתו הבא אחריו. אין להקליד רווחים לפני ערך מטיפוס char.

כאשר אתה מקליד מחרוזת, ()scanf מתחילה את ההצבה בתו הראשון שאינו תו רווח, ומפסיקה לייחס קלט למשתנה כשהיא נתקלת בתו הרווח הבא. התבונן בתוכנית הבאה:

```
main()
{
    char name[25];
    puts("Please enter your name: ");
    scanf("%s", name);
    puts(name);
}
```

שים לב שכאשר המשתנה הוא משתנה מחרוזת, אין משתמשים בתו &. אם תקליד Nancy ותקיש Enter, התווים יוצבו במשתנה name. אולם, אם תקליד Nancy Chesin (כמודגם בתרשים 5.10), ()scanf תציב את התווים החל מהתו הראשון שאינו רווח, ותפסיק את הצבת התווים בתו הרווח הבא. לפיכך, רק השם Nancy יוצב במשתנה, והפונקציה תתעלם משם המשפחה Chesin. בגלל אופן פעולתה של ()scanf, אם ברצונך להקליד מחרוזת שכוללת רווח, עליך להשתמש בפונקציה ()gets.

תרשים 5.10
()scanf מפסיקה לקרוא תווים לתוך משתנה מחרוזת ברגע שהיא נתקלת בתו רווח.



שימוש בפונקציה ()scanf

הבה נבחן עתה את הפונקציה ()scanf ביתר פירוט. עיין בתוכנית הבאה:

```
main()
{
        int count;
        printf("Please enter an integer number then press Enter:
");
        scanf("%d", &count);
        printf("Confirming your entry: %d ", count);
}
```

כאשר התוכנית מורצת, מתרחשים הדברים הבאים:

1. ההוראה ()printf מציגה הנחיה על-גבי המסך:

Please enter an integer number then press Enter:

מכיוון שלא כללנו קוד שורה-חדשה, הסמן נשאר בסופה של ההנחיה.

2. התוכנית ממתינה שהמשתמש יקליד ערך כלשהו.

3. אתה מקליד עיול כלשהו, ומקיש Enter.

4. ()scanf בוחנת את שטף הקלט וקובעת איזה תווים לקבל ומאיזה תווים להתעלם. אם היא נתקלת בתווים מספריים לפני תווים שאינם מספריים, היא ממשיכה לסרוק את השורה עד לתו הראשון שאינו תו מספרי, או עד לתו הרווח הראשון. התווים המספריים מומרים למספר ומאוחסנים בכתובתו של המשתנה. אם ()scanf נתקלת בתווים שאינם מספריים לפני תו מספרי כלשהו, הסריקה נפסקת והפונקציה מתעלמת מהתווים שהקלדת.

5. הסמן עובר לתחילתה של השורה הבאה במסך.

6. ההוראה ()printf מציגה את הערך של המשתנה count אחרי ההודעה confirming your entry.

אתה רשאי להשתמש במספר פונקציות ()scanf הדרוש לך כדי להזין את הנתונים. תוכנית 5.1 לדוגמה מזינה ארבעה משתנים - משתנה מטיפוס int, משתנה מטיפוס float, משתנה char ומחרוזת. שים לב שנעשה שימוש באופרטור הכתובת עבור כל המשתנים, למעט משתנה המחרוזת.



הערה

ניתן להשתמש ב-()scanf גם כדי להזין מספר משתנים בהוראה אחת. אולם, אם המשתמש אינו מקליד את הנתונים בדיוק באופן בו ()scanf מצפה לקבל אותם, הפונקציה עשויה להתעלם מהקלט כולו. לכן, כל עוד אתה לומד את שפת C, השתדל להימנע מהזנת מספר משתנים בפקודת ()scanf אחת. השתמש בפקודה נפרדת עבור כל פריט נתונים שעל המשתמש להקליד.

תוכנית 5.1: תוכנית שמזינה ארבעה משתנים

```
/* scanf3.c  */
main()
{
      int count;
      float amount;
      char letter;
      char name[15];
      puts("Enter an integer and press Enter");
      scanf("%d", &count);
      puts("Enter a float and press Enter");
      scanf("%f", &amount);
      puts("Enter a character and press Enter");
      scanf("%c", &letter);
      puts("Enter a string and press Enter");
      scanf("%s", name);
      printf("%d %6.2f %c %s", count , amount, letter, name);
}
```

ניתן להכריז על משתנה תווי כמשתייך לטיפוס הנתונים int, אולם עליך להזין אותו בעזרת מציין הפורמט c%. לא ניתן להזין אותו באמצעות המציין d% (אם כי באפשרותך להציג את ערך ה-ASCII של התו בעזרת המציין d% בפקודת ()printf).

השגת קלט מתאים

כפי שראינו, בגלל אופן הפעולה המיוחד של ()scanf, הפונקציה יכולה להתעלם מקלט. לכן, חיוני להציג הנחיה למשתמש לפני כל הוראת ()scanf.

לדוגמה, נניח שאתה כותב תוכנית המקבלת כקלט את סכומה של רכישה כלשהי:

```
main()
{
      float amount;
      puts("Please enter the amount of the sale: ");
      scanf("%f", &amount);
      printf("\nYour purchase was %f", amount);
}
```

מכיוון שהערך שיש להקליד הוא ערך כספי, משתמשים שיריצו את התוכנית עשויים לצרף לערך את סימן ה-$, כך: $45.65. סימן הדולר נכלל בשטף הקלט. אולם מכיוון שסימן הדולר אינו משתייך לטיפוס הנתונים float, ומכיוון שהוא התו הראשון שהפונקציה פוגשת בשטף, היא תתעלם מהקלט כולו. זו אינה דרך מוצלחת לעשות עסקים.

כדי להימנע מבעיה זו, יש לכלול הוראות מפורטות לפני פקודת scanf(), כדוגמת ההוראות הבאות:

```
puts("Please enter the amount of the sale. Type the\n");
puts("amount only. Do not type a dollar sign or use\n");
puts("commas between thousands. This is valid: 4567.87\n");
puts("This is not valid: $4,567.87\n");
puts(" Thank you\n");
```

עם זאת, ניתן לקבל קלט הכולל תווים שאינך מעוניין להציב במשתנה. זכור, בפונקציה printf(), מחרוזת הבקרה יכולה לכלול תווים נוספים, מלבד מצייני הפורמט. כל התווים המילוליים הכלולים במחרוזת הבקרה מודפסים, כשהערכים מוחדרים בעמדות של המציינים. באפשרותך לכלול תווים מילוליים גם במחרוזת הבקרה של הוראת scanf().

נניח שכתבת את הוראת ה-scanf() כך:

```
scanf("$%f", &amount);
```

סימן הדולר ($) במחרוזת הבקרה מורה ל-C לצפות לסימן $ כתו ראשון שאינו תו רווח בקלט. כאשר המשתמש מקליד $45.65, C מצפה כבר שסימן הדולר יהיה כלול בשטף. היא מתעלמת מהסימן במקום לעצור את קליטת הנתונים, ומאחסנת את התווים המספריים הבאים אחריו בכתובת של המשתנה amount.

יחד עם זאת, *חובה להקליד* כל מלל המופיע במחרוזת הבקרה לתוך שטף הקלט, ועליו להיות במקום המדויק יחסית למצייני הקלט. אם התווים אינם מוקלדים במקום בו הם אמורים להיות, הפונקציה מתעלמת מכל הקלט המוקלד מנקודה זו והלאה. לדוגמה, כאשר סימן הדולר כלול במציין הפורמט, הפונקציה תתעלם מהקלט אם לא תקליד את סימן הדולר כחלק מהעיול. לכן, אם החלטת להשתמש בתו המילולי, ודא שהמשתמש יודע שעליו להקליד אותו:

```
puts("Please enter the amount of the sale. Start the \n");
puts("number with a dollar sign, as in $4567.87\n"):
puts("Do not use commas between thousands.\n"):
```

סימן הדולר אינו מאוחסן כחלק מהמשתנה ואינו מוצג כאשר אתה משתמש במשתנה בפקודת printf(). הוא מופיע על המסך רק בעת הקלדת הנתונים, מכיוון שכל התווים

מופיעים על המסך בעת הקלדתם. אם ברצונך להציג סימן דולר יחד עם המספר בשלב מאוחר יותר בתוכנית, יש לבנות את הוראת ()printf כך:

```
printf("Your purchase was $%f", amount);
```

במקרה זה, סימן הדולר הוא פשוט עוד תו מילולי המוצג על ידי ()printf. הוא יוצג לפני המספר המוחדר במיקום של המצביע f%.

בגלל ההסתברות הגבוהה לבעיות הנובעות משימוש בתווים מילוליים בפונקציה ()scanf, מומלץ להימנע משימוש בהם, אלא אם הדבר חיוני או נדרש לצורך יישום מיוחד.

היזהר בשימוש בפונקציה ()scanf

במערכות מחשב רבות, תווים המוקלדים לתוך שטף מאוחסנים באזור מיוחד בזיכרון הנקרא חוצץ (buffer). אם ()scanf מפסיקה את פעולתה בטרם עת, תווים אחדים עשויים להישאר בחוצץ במקום להיות מוצבים במשתנים. עלול להיווצר מצב בו פעולת הקלט הבאה תתחיל לקרוא את התווים שנותרו בחוצץ מפעולת הקלט הקודמת, במקום את התווים שהמשתמש מקליד בתגובה להנחיית הקלט הבאה.

לכן יש לנקוט זהירות בשימוש בפונקציה ()scanf. מומלץ להזין כל פריט נתונים בנפרד, ולכלול הנחיות למשתמש המסבירות בבירור את טיפוס הקלט הנדרש.

הקלט בשפת התכנון ++C

קומפיילרים של ++C תומכים בפונקציות ()gets, ()getchar ו-()scanf, כפי שהכרת אותן בפרק זה. אולם, ++C כוללת פקודת קלט רב-שימושית נוספת, באמצעותה ניתן להזין את כל טיפוסי הנתונים. הפקודה cin, יחד עם אופרטור *החילוץ* <<, משמשת להשגת קלט מהמקלדת. ההוראות

```
int count;
cin >> count;
```

מזינות נתון מטיפוס int לתוך המשתנה count. בעבודה עם cin אין צורך להשתמש באופרטור הכתובת עבור משתנים מספריים ומשתנים מטיפוס char - די בשמו של המשתנה.

אין גם צורך להעביר מצייני פורמט לפונקציה. cin היא פקודת *העמסת-יתר*, כלומר - מסוגלת לקבוע כיצד לפעול בהתאם לנתונים שהיא מקבלת. בגלל תכונת העמסת-היתר, מעדיפים מרבית המתכנתים להשתמש ב-cin במקום ב-()scanf.

כדי להזין מספר ארגומנטים, יש להפריד בין שמות המשתנים באמצעות אופרטור החילוץ, כמו בדוגמה להלן:

```
cin >> amount >> count >> age >> name;
```

ייתכן שבהתחלה תתקשה לזכור אם להשתמש באופרטור החילוץ >> או באופרטור ההחדרה <<. תרשים 5.11 מציג שיטה קלה לזכירה. דמיין לעצמך שהאופרטורים הם חצים המופנים בכיוון המשתנים שברשימת הנתונים <<, או בכיוון הנגדי למשתנים שברשימת הנתונים >>.

תרשים 5.11 אופרטור ההחדרה ואופרטור החילוץ.

דמיין את הקלט והפלט מנקודת מבטו של המשתנה

cin>>variable;
>>>>>>>>>>>>

הנתונים נשלחים אל המשתנה מהמקלדת

cout<<variable;
<<<<<<<<<<<<<<

הנתונים נשלחים מהמשתנה אל עבר הצג

דמה את אופרטור החילוץ >> לחצים המופנים לכיוון המשתנים, ומסמלים נתונים העוברים אל המשתנים באמצעות הקלט. כאשר אתה משתמש ב-cin לצורך הזנת נתונים, השתמש באופרטור המצביע לכיוון המשתנים שלתוכם אתה מזין נתונים.

דמה את אופרטור ההחדרה << לחצים המופנים בכיוון הנגדי למשתנים, ומסמלים נתונים היוצאים מהמשתנים אל הצג. כאשר אתה משתמש ב-cout לצורך הצגת פלט, השתמש באופרטור המצביע לכיוון הנגדי למשתנים שמעבירים נתונים לצג.

משתנים שלא נקבע להם ערך

ייתכן מצב בו הכרזת על משתנה בתוכנית, אך לא הצבת בו ערך כלשהו. מצב כזה עשוי להיגרם בגלל הוראת ()scanf שנקטעה מוקדם מדי, או בגלל שפשוט שכחת לכתוב פקודת קלט להזנת הערך. אם אתה עושה שימוש במשתנה שלא נקבע לו ערך בתוך פקודת פלט, לא תופיע הודעת שגיאה, אבל הערך שיוצג על-גבי המסך יהיה חסר משמעות.

עם הפעלת המחשב, מתמלאות עמדות זיכרון שמערכת ההפעלה אינה משתמשת בהן בנתונים אקראיים. כאשר C מקצה עמדות זיכרון למשתנה מוכרז, היא אינה משנה את תכולתן של עמדות זיכרון אלה עד שמקצים למשתנה ערך התחלתי או עד שמזינים ערך כלשהו למשתנה. לכן, אם משגרים לפלט משתנה שלא הוקצה לו ערך, יופיעו בו הנתונים האקראיים המאוחסנים באותה עמדת זיכרון. אך המצב יכול להיות גרוע אף יותר, מכיוון שגם כאשר מנסים להשתמש במשתנה בחישוב כלשהו לא מופיעה הודעת שגיאה, אך תוצאת החישוב תהיה חסרת משמעות לחלוטין. אם אינך ער לכך, אתה עלול להניח שזוהי התוצאה הנכונה.

מתכנתים רבים נוהגים להקצות ערכים התחלתיים לכל המשתנים כדי למנוע את הבעיה. ניתן להקצות את הערך אפס למשתנים מספריים, ורווח למשתנים מטיפוס char ולמשתני מחרוזת, כך:

```
int count=0;
char initial=' ';
float rate=0.0;
```

ייתכן, כמובן, שהלוגיקה הפנימית בתוכנית שלך תכתיב ערכים התחלתיים אחרים.

בדרך זו, אם מציגים משתנה מבלי שהוזן לתוכו ערך או הוקצה לו ערך התחלתי, התוצאה המוצגת על המסך איננה חסרת משמעות. בהמשך תלמד כיצד לבחון את תוכנם של משתנים, כדי לוודא שהערכים המוצבים בהם תקפים. זכור, אם נכנס *זבל*, יוצא *זבל*.

טבלה 5.1 מסכמת את פקודות הקלט שלמדת בפרק זה.



הערה
מרבית התרגמנים (כמו התרגמן לשפת התכנות BASIC), וכן קומפיילרים אחדים, מקצים באופן אוטומטי אפסים למשתנים מספריים.

אלגוריתמים שימושיים לקלט

אלגוריתם הוא דרך לביצוע מטלה מסוימת. כאשר אתה לומד לתכנת, אתה לומד למעשה כיצד לפתח אלגוריתמים - כיצד לבצע מטלה בשפת התכנות C. נראה, לכאורה, שמספר האלגוריתמים האפשריים הוא בלתי מוגבל, אך למעשה, קיים גרעין יסודי של אלגוריתמים המשמשים לפיתוח 90 אחוז מהתוכניות. מרגע שתלמד להכיר את האלגוריתמים השימושיים ביותר, תוכל לעצב את התוכנית בקלות בעזרת השיטות שכבר למדת.

אחד האלגוריתמים האלה משמש להצבת ערך חדש במשתנה. ישנם מקרים בהם אתה מעוניין להזין ערך למשתנה שכבר מכיל ערך כלשהו. אם תשתמש, בפשטות, בשמו של המשתנה בפקודת קלט, יאבד הערך המקורי המאוחסן במשתנה. מה ניתן לעשות כאשר אינך רוצה לאבד את הערך המקורי? כאשר אתה רוצה, לדוגמה, להשוות את הערך הקודם לערך החדש, כפי שתלמד לעשות בפרק 8.

האלגוריתם משמש, בפשטות, להצבת הערך במשתנה אחר, כך:

```
cache=amount
```

בדוגמה זו, הערך המאוחסן במשתנה amount מוצב במשתנה cache. עד שתקצה או תזין ערך חדש למשתנה amount, יכילו שני המשתנים ערך זהה. המשתנה cache משמש כעמדת זיכרון נוחה לאחסון הערך עד שתזדקק לו שוב. כאשר תזדקק לערך זה, תקרא למשתנה המשמש כעמדת אחסון.

טבלה 5.1
סיכום פקודות הקלט.

פונקציה		הערות
gets()	טיפוס מחרוזת בלבד	שפת C ושפת ++C. משמשת להזנת מחרוזות, ובכלל זה גם מחרוזות הכוללות רווחים. יש להקיש Enter אחרי הקלדת המחרוזת.
getchar()	תווים יחידים	שפת C ושפת ++C. משמשת להזנת תו שהוכרז כמשתייך לטיפוס int או לטיפוס char. אין צורך להקיש Enter אחרי הקלדת התו. השימוש בפונקציה ()getchar נעשה ללא פרמטר, כמשתנה (לדוגמה, letter = getchar()). ניתן להשתמש בפונקציה עצמה כדי להשהות את הרצת התוכנית.
scanf()	כל טיפוסי הנתונים	שפת C ושפת ++C. זקוקה למצייני פורמט עבור כל פריט נתונים שיש להזין. אין להשתמש בפונקציה זו לצורך הזנת מחרוזות הכוללות רווחים. יש לוודא שהנתונים מוזנים ומוקצים למשתנים בסדר הנכון. ניתן להשתמש ב-()scanf כדי להזין מספר ארגומנטים בהוראה אחת. אין להקליד רווח לפני משתנה מטיפוס char.
cin	כל טיפוסי הנתונים	שפת ++C בלבד. אינה זקוקה למצייני פורמט או לאופרטורים של כתובות זיכרון. ניתן להזין בעזרתה מספר ארגומנטים, המופרדים בסימן >>.

תוכנית 5.2 מדגימה את תהליך הצבת ערכים חדשים במשתנים. (בפרק הבא תפגוש דוגמאות מעשיות יותר).

תוכנית 5.2: תוכנית המציבה ערך חדש במשתנה

```
/* storage.c */
main()
{
     int number, storage;
     puts("Enter the value of the number");
     scanf("%d", &number);
     storage=number;
     puts("Enter the value of the number");
     scanf("%d", &number);
     printf("The original value was %d\n", storage);
     printf("The new value is %d", number);
}
```

שורות אלה מציבות ערך במשתנה number:

```
puts("Enter the value of the number");
scanf("%d", &number);
```

הערך המאוחסן במשתנה number נשמר במשתנה storage באמצעות ההוראה:

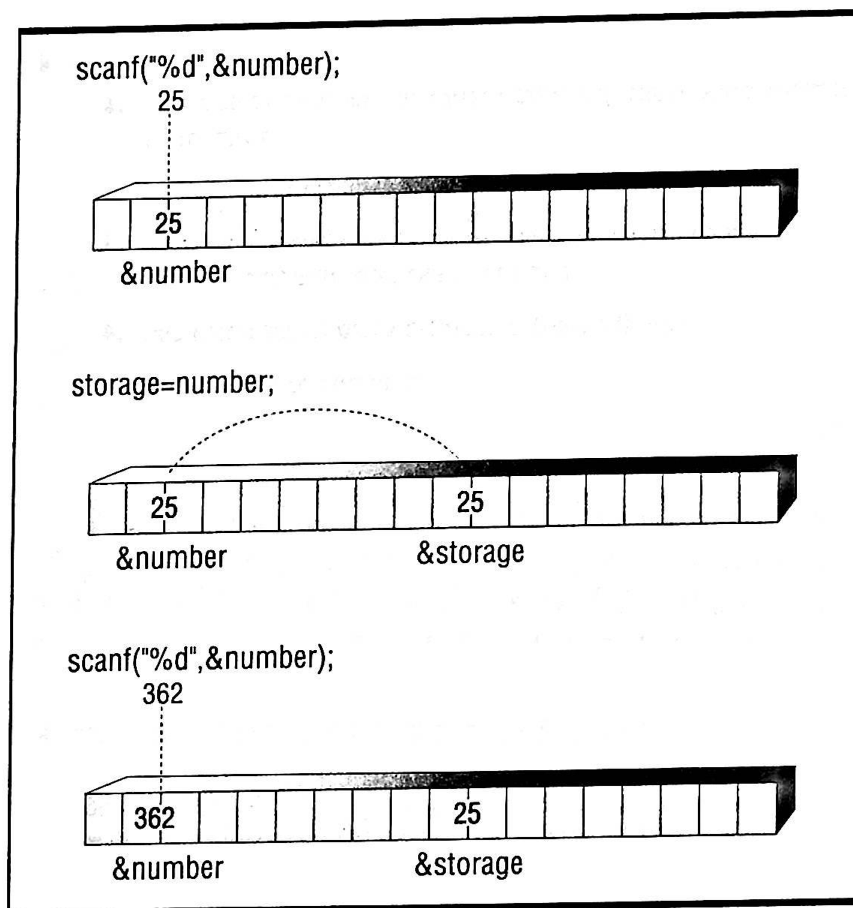
```
storage=number;
```

לאחר מכן מוצב ערך אחר במשתנה number, על ידי חזרה על ההוראות ששימשו להזנת הערך הראשון. ניתן להציג הן את הערך הישן והן את הערך החדש. הערך המקורי לא אבד, אלא אוחסן בעמדת זיכרון אחרת, כפי שמודגם בתרשים 5.12.

שאלות

1. מה פירוש הביטוי "נכנס זבל, יוצא זבל"?
2. מדוע יש להקיש Enter אחרי פקודת הקלט ()getchar?
3. מהן שתי הדרכים לשימוש בפונקצית הקלט ()getchar?
4. מהן אופרטור הכתובת?
5. מהם היתרונות בשימוש בפונקציה ()scanf להזנת נתונים?
6. מהם תווי המרה?

תרשים 5.12
תוצאות הצבה-מחדש של ערכים.



7. מתי תעדיף להשתמש ב-()gets במקום ב-()scanf להזנת מחרוזת?
8. מהם יתרונותיה וחסרונותיה של הוראת הקלט ()scanf עם מצייני פורמט?
9. האם יש צורך להקצות ערכים התחלתיים לכל המשתנים?
10. אילו בעיות עלולות להיווצר כאשר משתמשים ב-()scanf להזנת משתנה מטיפוס char?

תרגילים

1. כתוב תוכנית המזינה את שמך ומספר הטלפון שלך, ומציגה אותם בשורה אחת על-גבי המסך.

2. כתוב תוכנית המזינה מספר ומציגה את כתובת הזיכרון בה הוא מאוחסן.

3. כתוב תוכנית שמזינה שלושה מספרים ומציגה אותם בסדר הפוך - המספר האחרון שהוזן מוצג ראשון, וכך הלאה.

4. כתוב תוכנית שעושה שימוש ב-()getchar, ()gets ו-()scanf.

5. הסבר מה לקוי בתוכנית הבאה:

```
main()
{
	char initial;
	initial = gets();
	puts(initial);
}
```

6

סימני פעולה (אופרטורים)

מ שהו חסר! בפרק 5 למדת כיצד להזין נתונים למחשב. בפרק 4 למדת כיצד לשגר פלט אל הצג. עתה הגיע הזמן להכיר את מה שביניהם - שלב העיבוד.

שלב העיבוד בתוכנית ממיר את הנתונים שהזנת לתוכנית למידע, המהווה את פלט התוכנית. ההבדל בין נתונים לבין מידע הוא אומנם דק אך משמעותי. נתונים הם חומר גלם, תווים ומספרים שהם אומנם בעלי ערך, אך אינם מאורגנים בצורה שיוכלו לשמש כמוצר מוגמר. מידע הוא המוצר המוגמר; הוא התוצאה שכדי להשיג אותה כתבת את התוכנית.

תהליך ההמרה של נתונים למידע לובש צורות שונות. כשעובדים עם מספרים, התהליך כולל לעתים קרובות פעולות מתמטיות כאלה או אחרות. לדוגמה, ייתכן שאתה צריך להכפיל את סכום ההזמנה בשיעור מס הקנייה, ולהוסיף את המס לסכום ההזמנה. בתהליך זה, המודגם בתרשים 6.1, מעורבות שתי פעולות שניתן לתארן כך:

מס קנייה = סכום ההזמנה כפול שיעור המס

סך ההזמנה = סכום ההזמנה בתוספת מס הקנייה

תרשים 6.1
תהליך חישוב סך הזמנת קנייה.

Watson and Daughters
2867 Fifth Avenue
Abington,PA 19012

1	Widget	Green #203	$20.00
2	Melnabs	Large with ring	$80.00
1	Welblick	Standard	$10.00
		Subtotal	$110.00
		5% Sales Tax	5.50
		Total	$115.00

הכפל

110.0
× .05
5.50

חבר

110.0
+ 5.50
115.5

כדי לבצע את החישובים ההופכים נתונים למידע, אנו זקוקים ל*סימני פעולה (אופרטורים)*. אופרטור הוא סימן המורה למחשב כיצד לעבד את הנתונים. בפרק זה תכיר אופרטורים חשבוניים, אופרטורים של תוספות קבועות ואופרטורים של הקצאת ערכים.

הערות ++C

שפת C ושפת ++C משתמשות בסימני הפעולה המתמטיים בצורה דומה. עם זאת, בשפת ++C ניתן ליישם אופרטורים מסוימים לטיפוסי נתונים שונים, בזכות תכונת העמסת-היתר. בפרק 10 תלמד כיצד להשתמש באופרטור + בשפת ++C כדי לחבר בין שתי מחרוזות.

אופרטורים חשבוניים

כדי לבצע פעולות מתמטיות, משתמשים באופרטורים החשבוניים הבאים:

אופרטור	פעולה חשבונית
+	חיבור
-	חיסור
*	כפל
/	חילוק
%	שארית של חילוק מספרים שלמים

שים לב שהכוכבית משמשת לסימון פעולת הכפל (ולא התו X), ותו הלוכסן (/) מציין חילוק - אל תשתמש בתו הלוכסן ההפוך (\).

לעתים קרובות משתמשים באופרטורים בתוך הוראות המבצעות פעולות מתמטיות ומציבות את התוצאה במשתנה. בדוגמה שלנו, של חישוב סכום החשבונית, אין היגיון בהצבת ערך מילולי מדויק במשתנה המאחסן את מס הקנייה, כך:

```
sales_tax = 4500;
```

אלא יש לחשב את הערך. את המשתנה מציבים תמיד בצידו השמאלי של סימן השווה, ואת הפעולה החשבונית מצידו הימני:

```
sales_tax = amount * tax_rate;
price = cost + shipping + insurance;
per_unit = total / count;
```

הוראות אלה מורות לקומפיילר לבצע שלוש פעולות:

- להציב במשתנה sales_tax את הערך המתקבל מהכפלת הערך המוצב במשתנה amount בערך המוצב במשתנה tax_rate.
- להציב במשתנה price את הערך המתקבל מסכום שלושת הערכים cost פלוס shipping פלוס insurance.
- להציב במשתנה per_unit את התוצאה של חלוקת ערך המשתנה total בערך המשתנה count.

המחשב מבצע את הפעולה המתמטית הרשומה בצידה הימני של המשוואה, ומציב את הערך המתקבל במשתנה שבצידה השמאלי. ניתן להשתמש בכל שילוב של משתנים, קבועים וערכים מילוליים בצידו הימני של סימן השווה:

```
sales_tax = amount * 0.06;
price = 56.90 + shipping + 7.87;
per_unit = 156.65 / 12.50;
```

תוכנית 6.1 מדגימה את השימוש באופרטורים בתוכנית. התוכנית מזינה את שמותיהם וכתובותיהם של לקוחות, ואת סכומי ההזמנות שלהם. לאחר מכן היא מחשבת את דמי המשלוח (עשרה אחוז מסכום ההזמנה) ומס קנייה בשיעור שישה אחוזים. ההוראות ()puts ו-()printf מציגות חשבונית מלאה.

תוכנית 6.1: תוכנית לחישוב חשבונית ולהצגתה

```
/*invoice.c*/
#define tax_rate 0.06
#define shipping 0.10
main()
{
  char name[15], address[20], city[15], state[3], zip[6];
  float order, total, tax, ship;
  printf("Customer name: ");
  gets(name);
  printf("Street address: ");
  gets(address);
  printf("City: ");
  gets(city);
  printf("State: ");
  gets(state);
  printf("Zip: ");
  gets(zip);
  printf("Order amount: ");
  scanf("%f",&order);
  tax = order * tax_rate;
  ship = order * shipping;
  total = order + tax + ship;
  puts("\n\n\n\n");
  puts("                              INVOICE\n");
  printf("%s\n%s\n%s, %s  %s\n", name, address, city,
state,zip);
  printf("\t\t\t\t%-10s\t%10.2f\n", "Order:",order);
  printf("\t\t\t\t%-10s\t%10.2f\n", "Tax:",tax);
  printf("\t\t\t\t%-10s\t%10.2f\n", "Shipping:",ship);
  printf("\t\t\t\t\t\t_______________\n");
  printf("\t\t\t\t%-10s\t%10.2f", "Total:",total);
}
```

חילוק מספרים שלמים

כדי לחשב את השארית של חילוק מספרים שלמים, עושים שימוש באופרטור %. אם נשתמש באופרטור החילוק (/) לביצוע חילוק של מספרים שלמים, או של משתנים מטיפוס מספר שלם, התוצאה תהיה תמיד מספר שלם. לדוגמה, תוצאת חילוק 12 ב-5 (12/5) תהיה 2, ולא 2.4. השארית (0.4) היא מה שנותר אחרי חילוק מספרים שלמים.

לעתים אנו מעוניינים לדעת מהי השארית, אולם מכיוון שאנו עוסקים במספרים שלמים, לא ניתן להשתמש בערך 0.4, מכיוון שזהו מספר עשרוני, מטיפוס float. במקום זאת, אנו מאחסנים את השארית כמספר שלם. התוצאה של חילוק 12 ב-5 היא 2, והשארית היא 2, כלומר - השארית של פעולת החילוק 12/5, כשהיא מחושבת באמצעות הביטוי 5 % 12, היא 2. השארית היא תמיד מספר שלם.

לדוגמה, התבונן בתוכנית 6.2. תוכנית זו מחשבת כמה שטרות בני עשרים, עשרה, חמישה ואחד דולר דרושים כדי להחזיר עודף. הדבר החשוב ביותר שתוכנית זו מדגימה הוא האלגוריתם, הטכניקה העושה שימוש בחילוק מספרים שלמים כדי לבצע מטלה שהיתה עשויה לדרוש חישובים מורכבים יותר. בדומה לכל האלגוריתמים, גם אלגוריתם זה נראה פשוט למדי ברגע שקולטים את הרעיון הטמון בבסיסו.

תוכנית 6.2: תוכנית העושה שימוש באופרטור השארית לחישוב עודף

```
/*change.c*/
main()
{
int amount,twenties, tens, fives, ones, r20, r10;
printf("Enter the amount of change needed: ");
scanf("%d", &amount);
twenties= amount/20;
r20=amount % 20;  /* r20 represents remainder after twenties */
tens= r20/10;
r10=r20 % 10;     /* r10 represents remainder after tens */
fives= r10 / 5;
ones = r10 % 5;
putchar('\n');
printf("To make change for $%d give the following: \n", amount);
printf("%d twenty(ies)\n", twenties);
printf("%d ten(s)\n", tens);
printf("%d five(s)\n", fives);
printf("%d one(s)\n", ones);
}
```

אם תקליד את המספר 57, הפלט יהיה:

To make change for $57 give the following:

2 twenty(ies)

1 ten(s)

1 five(s)

2 one(s)

תרשים 6.2 מדגים את אופן פעולתה של התוכנית. מספר השטרות בני עשרים הדולר מחושב על ידי ההוראה twenties = amount/20. מכיוון שהמשתנים amount ו-twenties הם שניהם מספרים שלמים, התוצאה תהיה מספר שלם - מספר הפעמים ש-20 נכנס ב-amount. מבחינת המחשב, התוצאה היא רק מספר שיש לאחסנו בזיכרון. אך עבורנו, הערך של twenties הוא מידע, מכיוון שהוא מייצג את מספרם של השטרות בני עשרים הדולר בסכום העודף שיש להחזיר.

עתה, כשידוע לנו כמה שטרות בני עשרים דולר יש ב-amount (2), כיצד תקבע את מספר השטרות בני עשרה דולר? חשוב כיצד היית מבצע את המטלה הזו במציאות. לאחר מסירת השטרות בני עשרים הדולר, היית בודק מה הסכום הנותר, ולכמה שטרות בני עשרה דולר

תרשים 6.2
אופן פעולתה של תוכנית החזרת העודף.

	$57	
twenties=amount/20;	57/20=2	2 שטרות בני עשרים דולר
r20=amount%20;	57%20=17	אחרי השטרות בני 20 הדולר, נותרים 17
tens=r20/10;	17/10=1	שטר אחד בן עשרה דולרים
r10=r20%10;	17%10=7	אחרי השטר בן 10 דולרים, נותרים 7
fives=r10/5;	7/5=1	שטר אחד בן חמישה דולרים
ones=r10%5;	7%5=2	אחרי השטר בן חמישה הדולרים, נותרים שני שטרות בני דולר אחד

אתה זקוק. התוכנית עושה פעולה דומה. כדי לקבוע מהו הסכום שנותר, השתמש באופרטור השארית. ההוראה r20=amount % 20 מציבה במשתנה r20 את השארית שנותרה אחרי חלוקת המשתנה amount ב-20. עבורנו, שארית זו מייצגת את הסכום שנותר לאחר שהחזרנו כעודף את כל השטרות בני עשרים הדולר שניתן להחזיר. אחרי שמחזירים שני שטרות בני עשרים דולר, סך של $40, השארית היא 17.

לאחר מכן אנו חוזרים שוב על אותה התבנית - עבור השטרות בני עשרה דולר, tens, והשטרות בני חמישה דולר, fives. מספר השטרות בני דולר אחד שווה בדיוק לשארית הנותרת אחרי החילוק ב-fives.

אופרטורים וטיפוסי נתונים

בדרך כלל משתמשים באותו טיפוס נתונים (int או float) משני צדדיו של סימן השווה בפעולה חשבונית. אם עליך לחבר שני ערכים מטיפוס float, לדוגמה, תקצה את הסכום למשתנה מטיפוס float, כפי שמודגם בתוכנית הבאה:

```
main()
{
    float cost, shipping, total;
    cost = 56.09;
    shipping = 4.98;
    total = cost + shipping;
    printf("Your total bill is $%.2f", total);
}
```

הפלט של תוכנית זו הוא:

```
Your total bill is $61.07
```

ניתן גם להציב טיפוסי מספרים שונים משני צדדיה של הפעולה החשבונית. הערך שיוצג ייקבע על-פי טיפוס המשתנה שבצידה השמאלי של המשוואה. לדוגמה, עיין בגרסה שונה במקצת של התוכנית:

```
main()
{
    int total;
    float cost, shipping;
    cost = 56.09;
    shipping = 4.98;
    total = cost + shipping;
    printf("your total bill is $%d", total);
}
```

הפעולה החשבונית מחברת שני משתנים מטיפוס float (המשתנים cost ו-shipping) אולם מקצה את הסכום למשתנה מטיפוס int ששמו total. אם תחבר את הערכים במחשבון, הסכום יהיה 61.07. אך מכיוון שהמשתנה total משתייך לטיפוס הנתונים int, הסכום מומר ומוצג כמספר שלם. מציין הפורמט %d מציג את התוצאה כ-61.

שים לב שתחילה מבוצע החישוב המתמטי, ולאחר מכן התוכנית משלימה את הקצאת הערך למשתנה (ראה תרשים 6.3). אם הערכים היו מומרים למספרים שלמים לפני ביצוע החישוב, הסכום היה 60, 56 פלוס 4.

תרשים 6.3
החישוב המתמטי מבוצע לפני המרת הנתונים לטיפוס הנתונים של המשתנה.



ניתן גם לחבר שני מספרים שלמים ולהקצות את הסכום למשתנה מטיפוס float:

```
main()
{
    int cost, shipping;
    float total;
    cost = 56;
    shipping = 4;
    total = cost + shipping;
    printf("Your total bill is $%.2f", total);
}
```

במקרה זה יוצג מספר מטיפוס float, בגלל המציין %f וטיפוס הנתונים של המשתנה. אך מכיוון שהמספרים שסוכמו הם מספרים שלמים, הספרות העשרוניות יהיו אפסים - 60.00.

ניתן גם לשלב מספרים שלמים ומספרים עשרוניים בצידו הימני של סימן השווה. גם במקרה זה, בפעולות חיבור וחיסור, אופן הצגת התוצאה תלוי בטיפוס הנתונים של המשתנה בו מציבים אותה. לדוגמה, עיין בהוראות הבאות:

```
main()
{
    int shipping;
    float total, cost;
    cost = 56.09;
    shipping = 4;
    total = cost + shipping;
    printf("Your total bill is $%.2f", total);
}
```

הפעולה החשבונית מחברת ערך מטיפוס float עם מספר שלם, והתוצאה תופיע כמספר מטיפוס float, 60.09.

כללים אלה תקפים גם בחילוק. אם מחלקים שני מספרים (ערכים מילוליים) ומקצים את המנה למשתנה מטיפוס float, חייב לפחות אחד המספרים לכלול נקודה עשרונית, אם ברצונך להציג את הפלט כמספר עשרוני. תהליך זה מודגם בתוכנית המוצגת בתרשים 6.4. בדוגמה זו מבוצע חישוב זהה שלוש פעמים, והתוצאה מוצגת תמיד במשתנה מטיפוס float. עם זאת, בחישוב הראשון לא כולל אף אחד מהמספרים נקודה עשרונית, ולכן התוצאה מוצגת בצורה שגויה, עם אפסים בספרות העשרוניות.

תרשים 6.4
חילוק ערכים מילוליים.

```
main()
{
    float c;                    פלט
    c=12/5;                     Output
    printf("%f\n",c);           2.000000
    c=12.0/5;
    printf("%f\n",c);           2.400000
    c=12/5.0;
    printf("%f\n",c);           2.400000
}
```

ביטויים

ביטוי הוא חלקה הימני של משוואה ללא משתנה מצידה השמאלי. ביטויים משמשים בהוראות פלט, כמו בדוגמה הבאה:

```
main()
{
	int count;
	count = 5;
	printf("The number is %d", count + 19);
}
```

תוכנית זו מציגה את הפלט:

```
The number is 24
```

הביטוי הכלול בהוראת ה-printf() הוא count + 19. כאשר מתבצעת קריאה לפונקציה printf(), מחושב תחילה ערכו של הביטוי. תוצאת הביטוי מוצגת על ידי המציין %d.

ביטוי אינו משנה את ערכו של המשתנה. בדוגמה שלנו, אחרי ביצוע החישוב count + 19 והצגת התוצאה 24, ערכו של המשתנה count נותר עדיין 5. זכור, ביטוי הוא רק חלקה הימני של המשוואה. הערך שחושב אינו מוקצה למשתנה כלשהו.

ביטויים יכולים לכלול כל שילוב של קבועים, משתנים וערכים מילוליים, וכן אופרטורים:

```
printf("%d", count + number);
printf("%d", 16 - 4);
printf("%f", amount * tax_rate);
```

ככלל, ניתן להשתמש בביטוי בכל מקום בו ניתן להשתמש במשתנה. עם זאת, יש לשקול היטב בכל מקרה אם להשתמש בביטוי או להציב את התוצאה במשתנה. לדוגמה, עיין בתוכנית הבאה:

```
main()
{
	float cost, shipping;
	printf("Enter the cost of the item: ");
	scanf("%f", &cost);
	printf("Enter the shipping charge: ");
	scanf("%f", &shipping);
	printf("The total due is %f", cost + shipping);
}
```

ההוראה:

```
printf("The total due is %f", cost + shipping);
```

מציגה פלט זהה לזה שמציגות ההוראות:

```
total = cost + shipping; /*total must be declared as a float*/
printf("Amount due equals %f", total);
```

שתי ההוראות מציגות את הסכום שיש לשלם. במקרה הראשון, העושה שימוש בביטוי, אין צורך להכריז על משתנה שיאחסן את תוצאות החישוב, ואין גם צורך להקליד משוואה. החישוב המתמטי מתבצע והתוצאה מוצגת מייד באמצעות הוראת printf(). עם זאת, תוצאת החישוב אינה מאוחסנת בזיכרון המחשב. אם בשלב מאוחר יותר תרצה להציג את הסכום שיש לשלם, יהיה עליך להשתמש שוב בביטוי cost + shipping.

לעומת זאת, כאשר מקצים את התוצאה למשתנה, כמו בסדרת ההוראות השנייה, יש להכריז על משתנה ולכתוב את המשוואה. אולם הפעם, תוצאות החישוב מאוחסנות בזיכרון. אם תרצה להשתמש פעם נוספת בערך שחישבת, כל שיהיה עליך לעשות הוא להקליד את שמו של המשתנה, במקום את הביטוי כולו.

אם עליך להשתמש בתוצאה של חישוב מתמטי יותר מפעם אחת, כדאי להקצות אותה למשתנה ולכתוב את המשוואה. השתמש בביטוי כאשר אתה זקוק לתוצאה פעם אחת בלבד.

סדר קדימויות

כאשר המחשב נתקל במשוואה, הוא אינו מבצע את הפעולות החשבוניות באופן אוטומטי, לפי סדר הופעתן משמאל לימין. המחשב סורק תחילה את השורה, ומבצע את הפעולות על-פי סדר הקדימויות של פעולות החשבון. סדר *קדימויות* פירושו שפעולות מסוימות מתבצעות לפני פעולות אחרות, אפילו אם הן מופיעות במשוואה אחריהן. הסדר קובע שכפל וחילוק מתבצעים תחילה, ולאחר מכן חיבור וחיסור. דרך קלה לזכור את סדר הקדימויות היא להיעזר במשפט מנמוני, כמו: My Dear Aunt Sally (M = Multiplication - כפל, D = Division - חילוק, A = Addition - חיבור, ו-S = Subtraction - חיסור).

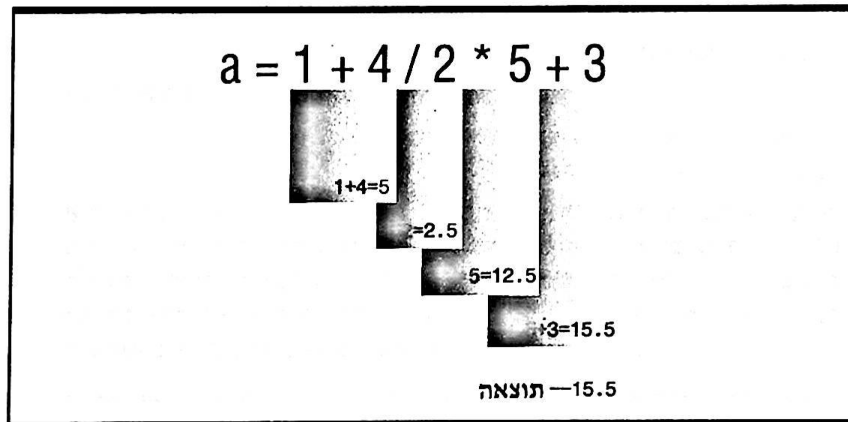
המחשב סורק את המשוואה ומבצע תחילה פעולות כפל וחילוק, על-פי הסדר בו הן מופיעות. פעולת חילוק המופיעה לפני פעולת כפל - תבוצע תחילה. לשתיהן סדר קדימויות זהה. המחשב מקצה עמדות זיכרון זמניות כדי לאחסן את התוצאות, ואז חוזר לתחילת המשוואה ומבצע את פעולות החיבור והחיסור, על-פי סדר הופעתן.

עיין במשוואה הפשוטה הבאה:

```
a= 1 + 4 / 2 * 5 + 3;
```

אם נבצע את הפעולות על-פי סדר הופעתן משמאל לימין, כמו במחשבון, התוצאה תהיה 15.5 (ראה תרשים 6.5). המחשב לעומת זאת זוכר את המשפט My Dear Aunt Sally, ותוצאת החישוב שלו תהיה 14. (תרשים 6.6 מדגים כיצד מתבצע החישוב). עליך לוודא שפעולות החשבון בתוכנית שלך מתבצעות בסדר הרצוי.

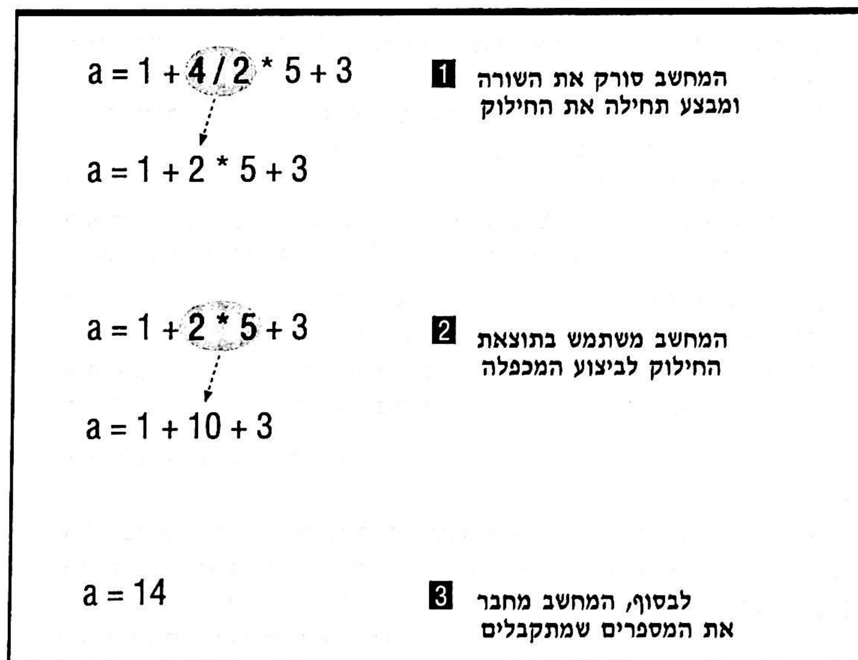
תרשים 6.5
ביצוע פעולות חשבון משמאל לימין, כמו במחשבון.



לדוגמה, השגיאה הנפוצה ביותר שעושים מתכנתים מתחילים קשורה בחישוב ממוצעים. תוכנית 6.3 מדגימה תוכנית שמנסה להזין שלושה מספרים ולחשב את ערכם הממוצע. השתמשנו במלה "מנסה", מכיוון שהממוצע שמחשבת התוכנית יהיה תמיד שגוי. האם אתה רואה מדוע? השגיאה טמונה בשורה הזו:

```
average = number_1 + number_2 + number_3 /3;
```

תרשים 6.6
ביצוע פעולות החשבון על-פי סדר הקדימויות.



המחשב מבצע את הפעולות בסדר הבא:

1. מחלק את number_3 ב-3.

2. מוסיף את המנה ל-number_1 ול-number_2.

תוכנית 6.3: תוכנית שגויה לחישוב ממוצע של שלושה מספרים

```
/*average.c*/
main()
{
     float number_1, number_2, number_3, average;
     printf("Enter the first number: ");
     scanf("%f", &number_1);
     printf("Enter the second number: ");
     scanf("%f", &number_2);
     printf("Enter the third number: ");
     scanf("%f", &number_3);
     average = number_1 + number_2 + number_3 / 3;
     printf("The average is %f", average);
}
```

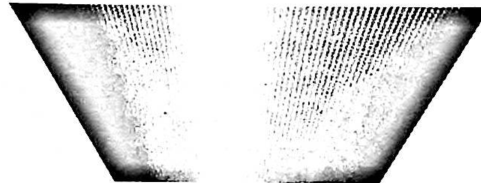
אם תקליד את הערך 100 עבור כל אחד מהמספרים, הממוצע יוצג כ-233.33. כדי לכתוב את ההוראה כהלכה, השתמש בצורת הכתיבה הבאה:

```
average = (number_1 + number_2 + number_3) /3;
```

הסוגריים משנים את סדר הקדימויות. כאשר המחשב סורק את המשוואה, הוא מבצע תחילה את הפעולות הנתונות בסוגריים, ואז חוזר לתחילת השורה ומבצע את הפעולות הנותרות על-פי סדר הקדימויות. במקרה זה, המחשב מסכם תחילה את שלושת המספרים, ואז מחלק את הסכום בשלוש, כמודגם בתרשים 6.7. זהו זה! התשובה הנכונה.

תרשים 6.7
הנוסחה הנכונה לחישוב הממוצע.

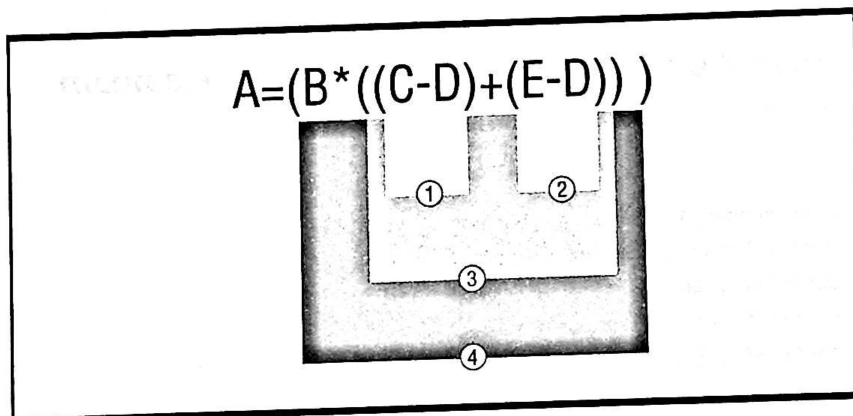
average = (number_1 + number_2 + number_3) / 3



average = (the sum of the numbers) / 3

כדי לשלוט בסדר הפעולות אף יותר, ניתן להשתמש במספר רמות של סוגריים, כמודגם בתרשים 6.8. המחשב מבצע תחילה את הפעולות הנתונות בסוגריים הפנימיים ביותר, ומתקדם משם לכיוון הרמות החיצוניות.

תרשים 6.8
מספר רמות של סוגריים.



נתבונן בדוגמה. נניח שאתה משלם לעובדיך תעריף כפול עבור כל שעת עבודה מעל 40 שעות עבודה בשבוע. בהנחה שכל העובדים עובדים לפחות 40 שעות, או מקבלים תשלום עבור 40 שעות לכל הפחות, אנו זקוקים לחלקים הבאים כדי לבצע את החישוב:

```
40 * rate     /* normal weekly salary
hours-40      /* overtime hours
rate * 2      /* double-time rate
```

על המשוואה להכפיל את מספר שעות העבודה הנוספות בתעריף הכפול, ולהוסיף את תוצאת המכפלה למשכורת השבועית הרגילה. אם מתעלמים מסדר הקדימויות, ניתן לכתוב את המשוואה כך:

```
total = 40 * rate + hours - 40 * rate * 2
```

נניח שעובד כלשהו עבד 48 שעות, לפי תעריף של $10 לשעה. ללא סוגריים, תחושב המשוואה כך:

פעולה	תוצאה
40 * rate	400
40 * rate	400
400 * 2	800
400 + 48	448
448-800	-352

ודאי שתשלום בסך מינוס $352 לא נראה כתשלום הנכון. כדי לכתוב את המשוואה כהלכה, יש להשתמש בסוגריים:

```
total = (40 * rate) + ((hours - 40) * (rate * 2))
```

המשוואה מחולקת עתה לשני חלקים לוגיים (תוכנית 6.4). הפריטים בתוך הסוגריים הראשונים מייצגים שעות עבודה רגילות, ואילו הפריטים בתוך הסוגריים השניים מייצגים שעות עבודה נוספות. הקטע הזה עצמו מכיל שתי רמות של סוגריים. קומפיילר C יבצע תחילה את הפעולות שבסוגריים הפנימיים, ולאחריהן את הפעולה שבסוגריים החיצוניים. כלומר, מתבצעות חמש פעולות:

400	40 * rate
8	hours - 40
20	rate * 2
160	8 * 20
560	400+160

תוכנית 6.4: חישוב משכורת כולל תשלום עבור שעות נוספות

```
/*payroll1.c*/
main()
{
    float rate, hours, total;
    printf("Enter your hourly rate of pay: ");
    scanf("%f", &rate);
    printf("Enter the number of hours you worked: ");
    scanf("%f", &hours);
    total = (40 * rate) + ((hours - 40) * (rate * 2));
    printf("Your salary is %f", total);
}
```

אם אתה מתקשה לעבוד עם סוגריים, באפשרותך לפרק את המשוואה למשתנים נפרדים. תוכנית 6.5 מחשבת אף היא משכורת כולל תשלום עבור שעות נוספות, אולם החישוב מתבצע בסדרה של פעולות, במקום במשוואה ארוכה אחת. מכיוון שתוצאת כל פעולה מוצבת במשתנה, ניתן להציג כל תוצאה בנפרד, ובכך לעשות את הפלט לברור יותר. טכניקה זו מעניקה לך בקרה טובה יותר על התהליך, ומקלה על איתור שגיאות.

תוכנית 6.5: תוכנית עם פעולות נפרדות

```
/*payroll1.c*/
main()
{
    float rate, hours, total, regular, extra, d_time, overtime;
    printf("Enter your hourly rate of pay: ");
    scanf("%f", &rate);
    printf("Enter the number of hours you worked: ");
    scanf("%f", &hours);
    regular = 40 * rate;
    extra = hours - 40;
    d_time=rate * 2;
    overtime = extra * d_time;
    total = regular + overtime;
    printf("Your regular weekly salary is %.2f\n", regular);
    printf("You worked %.2f overtime hours\n", extra);
    printf("Your overtime pay is %.2f\n", overtime);
    printf("Your total pay is %.2f\n", total);
}
```

אלגוריתמים שימושיים לעיבוד נתונים

חלק מהאלגוריתמים השימושיים ביותר בתכנות כוללים פעולות חשבוניות. השימוש בכמה מהאלגוריתמים האלה הוא כה נפוץ, עד שמתכנתים רבים כלל לא חושבים עליהם כעל אלגוריתמים. מתוכם, שני האלגוריתמים החשובים ביותר הם המונים (counters) והאוגרים (accumulators).

תרשים 6.9
אלגוריתם המונה.



מונים

מונה הוא משתנה המתחיל בערך התחלתי כלשהו, הגדל באחד בכל פעם שמשהו מתרחש. האלגוריתם עבור מונה הוא:

```
variable = variable + 1
```

אין ספק שכל מורה למתמטיקה יזדעק למראה המשוואה הזו ויטען שאי אפשר להשתמש באותו משתנה משני צדדיה של המשוואה. אולם בתכנות מחשבים זוהי פעולה תקפה לחלוטין.

המחשב מחשב תחילה את הביטוי שבצידה הימני של המשוואה, ואת התוצאה מציב במשתנה שבצידה השמאלי (ראה תרשים 6.9). בכל רגע נתון מוצב במשתנה ערך אחד בלבד. ניתן להבין את המונה כך:

הערך החדש של המשתנה שווה לערך הקודם פלוס 1.

הבה נבחן את שלבי אלגוריתם המונה - כפי שמודגם בתרשים 6.10. נתחיל עם משתנה ששמו count ונקצה לו ערך התחלתי אפס:

```
int count;
count=0;
```

כעת ניישם את האלגוריתם:

```
count = count + 1
```

המחשב מבצע את ההוראה כך:

```
count = 0 + 1
```

הערך הנוכחי של count, (0), מתווסף לערך המילולי 1. תוצאת הפעולה, 1, מוצבת במשתנה count כערך חדש.

הבה נבצע את ההוראה הזו פעם נוספת:

```
count = count + 1
```

המחשב מבצע את הפעולה הבאה:

```
count = 1 + 1
```

הערך הנוכחי של count, (1), מתווסף לערך המילולי 1. התוצאה, 2, מוצבת במשתנה count. בכל פעם שמבוצעת ההוראה, ערכו של count גדל באחד.

כמובן שניתן למנות בעזרת כל מספר ולהציב כל מספר כערך התחלתי. אם נקצה ל-count את הערך 1 כערך התחלתי, ההוראה

```
count=count + 2
```

תרשים 6.10
ביצוע אלגוריתם המונה.

ערכו של המשתנה count אחרי ההוראה	החישוב שמתבצע בפועל	הוראה
0		count=0;
1	count=0+1	count=count+1;
2	count=1+1	count=count+1;
3	count=2+1	count=count+1;
4	count=3+1	count=count+1;

תגדיל את ערכו של count לאורך ציר המספרים האי-זוגיים - 1, 3, 5, 7 וכן הלאה. ניתן למנות בדילוגים של 5 עם הביטוי count=count+5, או בדילוגים של 10 עם הביטוי count=count+10, או בדילוגים של כל מספר אחר.

כדי למנות כלפי מטה (להקטין את הערך של count) יש לשנות את האלגוריתם כך:

```
variable=variable-1
```

בכל פעם שהפעולה מתבצעת, הערך של המשתנה קטן באחד.

אופרטורים של תוספות קבועות

השימוש במונים הוא כה נפוץ, עד ששפת C כוללת אופרטורים מיוחדים המשמשים לביצוע תוספת קבועה לערך של משתנה. האופרטור ++variable מוסיף 1 לערך של המשתנה לפני ביצוע ההוראה. האופרטור מבצע למעשה את הפעולה

```
variable = variable + 1;
```

לדוגמה, נבחן את התוכנית הבאה:

```
/*count.c*/
main()
{
    int count=0;
    printf("The first count is %d\n", count);
    printf("The second count is %d\n", ++count);
    printf("The last value of count was %d\n", count);
}
```

הפלט של התוכנית הוא:

The first count is 0

The second count is 1

The last value of count was 1

לפני שהקומפיילר מבצע את הוראת ()printf השנייה, הוא מגדיל את הערך של count ב-1, כאילו כתבת את ההוראות כך:

```
count=count+1;
printf("The second count is %d\n", count);
```

על ידי שימוש באופרטור התוספת הקבועה ניתן לבצע את אלגוריתם המונה מבלי לכתוב הוראה נפרדת.

אופרטור התוספת משנה בפועל את ערכו של המשתנה. ודא שאתה מבין את ההבדל בין אופרטור התוספת ++count לבין הביטוי count+1 בשורה הבאה:

```
printf("The second count is %d\n", count+1);
```

הביטוי count+1 אינו משנה את ערכו של המשתנה count, אלא רק מציג את הערך אחרי התוספת. תוכנית 6.6 עושה שימוש בביטוי זה.

תוכנית 6.6: שימוש בביטוי במקום באופרטור התוספת הקבועה.

```
main()
{
     int count=0;
     printf("The first count is %d\n", count);
     printf("The second count is %d\n", count+1);
     printf("The last value of count was %d\n", count);
}
```

הפלט של התוכנית הוא:

```
The first count is 0
The second count is 1
The last value of count was 0
```

הביטוי count+1 מחושב כ-1, אך אינו משנה את הערך של המשתנה count. כאשר המשתנה מוצג על ידי הוראת ()printf השלישית, הערך שלו הוא עדיין הערך המקורי.

ניתן להשתמש באופרטור התוספת הקבועה בתוך הוראות ()printf, כמו בביטוי בצידה השמאלי של משוואה. ההוראה הבאה, לדוגמה, מגדילה את הערך של המשתנה count, ואז מציבה את הערך החדש במשתנה number:

```
number = ++count;
```

ביטוי זה מקביל לשתי ההוראות הבאות:

```
count = count + 1;
number = count;
```

ניתן אפילו להשתמש באופרטור ++, יחד עם שם של משתנה, כהוראה בפני עצמה:

```
++number;
```

אם מציבים את האופרטור ++ בסופו של שם המשתנה, הגדלת הערך מתבצעת אחרי סיום ביצוע ההוראה. הכתיב הוא:

```
variable++
```

התבונן בגרסה שונה של התוכנית הקודמת:

```
main()
{
      int count=0;
      printf("The first count is %d\n", count);
      printf("The second count is %d\n", count++);
      printf("The last value of count was %d\n", count);
}
```

האופרטור ++count מגדיל את הערך של המשתנה אחרי ביצוע הוראת ()printf השנייה. הוראת ()printf השנייה מציגה לפיכך את הערך המקורי, כפי שניתן לראות בפלט הבא:

```
The first count is 0
The second count is 0
The last value of count was 1
```

הערך של המשתנה הוא 0 כאשר מתבצעות הוראות ()printf הראשונה והשנייה. הערך גדל ל-1 לפני שמתבצעת הוראת ()printf השלישית.

ניתן להשתמש באופרטור התוספת כדי להקצות מחדש את הערך המקורי של משתנה ולהגדילו, במהלך אחד, כמו בתוכנית הבאה:

```
main()
{
      int number, storage;
       puts("Enter the value of the number");
      scanf("%d, &number);
      storage=number++;
      printf("The original value was %d\n", storage);
      printf("The new value is %d", number);
}
```

הפקודה:

```
storage=number++
```

מקצה תחילה למשתנה storage את הערך של המשתנה number, ואז מגדילה את הערך של number באחד.

האופרטורים להפחתה קבועה פועלים באופן דומה, אולם מקטינים את הערך של המשתנה באחד. הם נכתבים כך:

--variable	מפחית 1 מערכו של המשתנה לפני ביצוע ההוראה.
variable--	מפחית 1 מערכו של המשתנה אחרי ביצוע הפעולה.

אוגרים

גם אוגר מגדיל את ערכו של משתנה, אולם במקום להגדיל תמיד את הערך במספר קבוע, הגידול משתנה - כלומר, יכול להשתנות בכל פעולה. צורת הכתיב הכללית היא:

```
variable = variable + other_variable
```

אלגוריתם זה נקרא אוגר מכיוון שהערך ממשיך להיאגר. לדוגמה, התבונן בהוראות הבאות:

```
int total, number;
total = 0;
scanf("%d", &number);
total = total + number;
```

נניח שאתה מזין את הערך 10 כערכו של המשתנה number. אחרי ביצוע ההוראה

```
total = total + number;
```

הערך של total הוא 10. המחשב ביצע את ההוראה על הערכים הבאים:

```
total = 0 + 10;
```

כעת, נניח שאתה מבצע את הוראות ()scanf והאוגר פעם נוספת, ומזין את הערך 15:

```
scanf("%d", &number);
total = total + number;
```

במקרה זה, המחשב יבצע את ההוראות תוך שימוש בערכים:

```
total = 10 + 15;
```

אתה אוגר את הערכים של המשתנה number.

תוכנית 6.7 מקבלת קלט של שלושה מספרים ומחשבת את הממוצע שלהם. ניתן היה להשתמש בביטוי פשוט כדי לחשב את הממוצע:

```
average = (A+B+C)/3
```

השתמשנו באוגר כדי לסכם את המספרים ובמונה כדי למנות את העיולים - לצורך הדגמת אופן השימוש בהם. בפרק 9 תלמד כיצד להשתמש באלגוריתמים אלה באופן יעיל ביותר.

תוכנית 6.7: שימוש במונה ובאוגר כדי לחשב את הממוצע של שלושה מספרים

```
    /*average1.c*/
    main()
{
    float number, total, count, average;
    total = 0.0;
    count = 0.0;
    printf("Enter the first number: ");
    scanf("%f", &number);
    total += number;
    ++count;
    printf("Enter the second number: ");
    scanf("%f", &number);
    total += number;
    ++count;
    printf("Enter the third number: ");
    scanf("%f", &number);
    total += number;
    ++count;
    average = total/ count;
    printf("The average is %f", average);
}
```

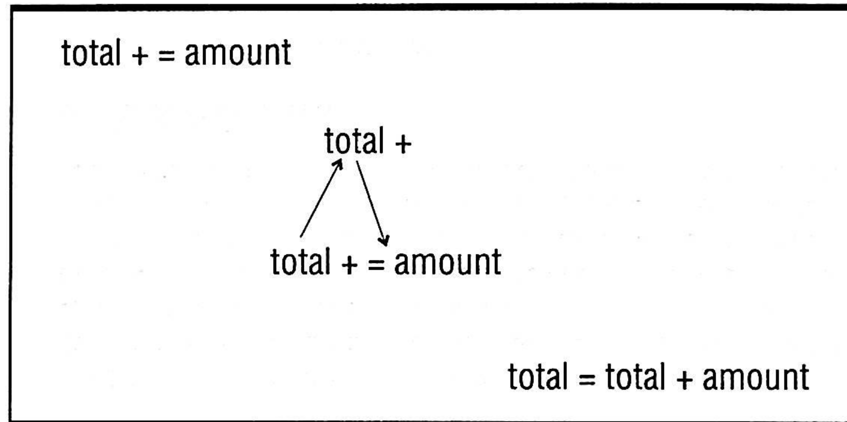
אופרטורים של הקצאת ערכים

האופרטורים המשמשים להקצאת ערכים הם למעשה אוגרים הרשומים בכתיב מקוצר:

אופרטור	דוגמה	מקביל ל-
+=	total += amount	total = total + amount
-=	total -= discount	total = total - discount
*=	total *= tax_rate	total = total * tax_rate
/=	total /= count	total = total / count
%=	total %= count	total = total % count

כל אחד מהאופרטורים מבצע פעולה תוך שימוש במשתנה בו מוצב הערך בתור הגורם המשותף. כדי להמחיש את אופן פעולתם, התבונן בתרשים 6.11. דמיין לעצמך שעותק בלתי נראה של המשתנה וסימן הפעולה שמשמאל לתו השווה - עוברים לצידה הימני של

תרשים 6.11
האופרטור של הקצאת ערכים



המשוואה. ההוראה

```
total *= rate
```

זהה להוראה

```
total = total * rate
```

הקצאת ערכים התחלתיים

חובה להקצות ערכים התחלתיים לכל המונים והאוגרים. זכור, שני האלגוריתמים מוסיפים ערך כלשהו לערך הנוכחי (או מפחיתים ממנו). אם לא מקצים למשתנה ערך התחלתי, המספר מתווסף לתוכנה האקראי, הבלתי ידוע, של כתובת המשתנה. הקצאת ערך התחלתי מנקה את כתובת המשתנה, בדומה להקשה על מקש Clear במחשבון.

לדוגמה, התבונן שוב בתוכנית 6.7, המחשבת את הממוצע של שלושה מספרים. נניח שכתובתו של המשתנה total מכילה את הערך האקראי 1827. אם התוכנית אינה כוללת את פקודת ההקצאה

```
total = 0;
```

ואתה מקליד את המספר 75, האוגר הראשון יבצע את הפעולה החשבונית כך:

```
total = 1827 + 75
```

הקצאת ערך התחלתי אפס למשתנה מבטיחה שהפעולה החשבונית תתבצע כהלכה, כך:

```
total = 0 + 75
```

עיצוב התוכנית

עתה, לאחר שלמדת כיצד לבצע פעולות מתמטיות בעזרת אופרטורים, הלוגיקה של התוכנית עשויה להיות מתוחכמת יותר. עליך להקפיד ולוודא שהתוכנית אכן משיגה את התוצאות הנכונות. הבה נבחן מספר דוגמאות.

היזהר משגיאות לוגיות

בתחילת הפרק (תוכנית 6.4) הצגנו כדוגמה תוכנית שמחשבת שכר שבועי. ניתן לקמפל את התוכנית ללא שגיאות ולהריץ אותה מספר רב של פעמים - לפני שתתגלה העובדה שהתוכנית כוללת פגם משמעותי - היא מניחה שכל אחד מהעובדים עובד לפחות 40 שעות בשבוע, או שהוא מקבל תשלום עבור 40 שעות לכל הפחות. אם ההנחות האלה נכונות, אין כל בעיה. אך מה בדבר עובדים שצברו פחות מ-40 שעות עבודה, ושזכאים לתשלום רק עבור שעות העבודה בפועל? אם נזין לתוכנית מספר הקטן מ-40 שעות, החישוב לא יהיה מדויק עבור עובדים אלה. הבעיה טמונה בחישוב:

```
total = (40 * rate) + ((hours - 40) * (rate * 2));
```

ראשית, גם במקרה שעובד צבר פחות מ-40 שעות, מספר השעות המוכפל בתעריף, בביטוי rate * 40, יהיה עדיין 40. שנית, התוצאה של הביטוי hours - 40 תהיה מספר שלילי. ערך שלילי זה יוכפל בתעריף הכפול ויתווסף לשכר עבור שעות עבודה רגילות. כאשר מוסיפים מספר שלילי, מבצעים למעשה פעולת חיסור. לפיכך, העובד יפסיד שכר בתעריף כפול עבור כל שעת עבודה שעבד פחות מ-40 שעות.

בפרק מאוחר יותר נפתור את הבעיה הזו. בשלב זה, חשוב יותר להבין את הבעיה מאשר למצוא את הפתרון.

חפש תבניות חוזרות

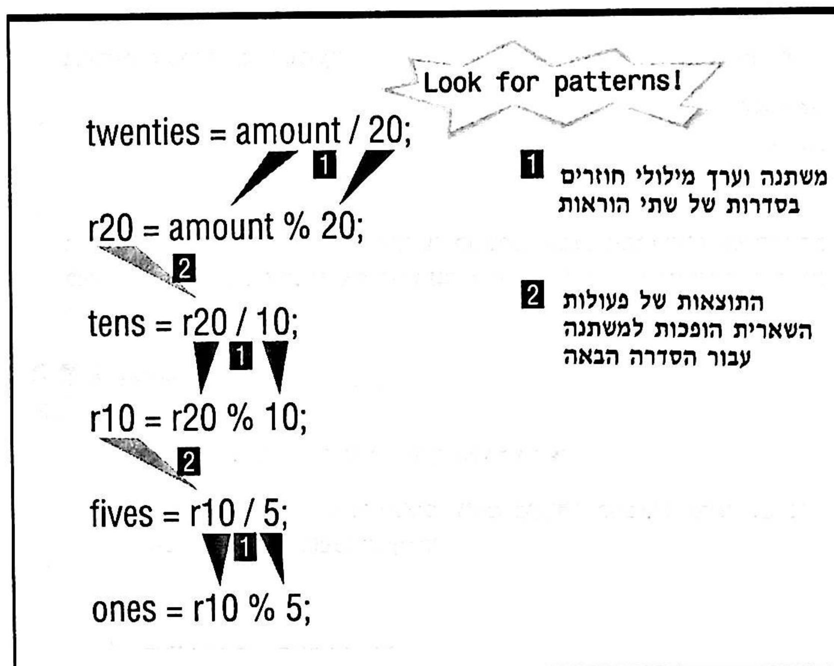
תוך כדי עבודת התכנות, נסה לאתר תבניות חוזרות. חפש הוראות שניתן להשתמש בהן יותר מפעם אחת, או הוראות שחוזרות בשינויים קטנים בלבד. הבחנה בתבניות מקלה על הבנת האופן בו פועלת התוכנית, ומכאן - מקלה גם על כתיבתה של תוכנית דומה בהזדמנות אחרת.

לדוגמה, תוכנית 6.2, שעושה שימוש בחילוק מספרים שלמים לחישוב עודף, כוללת תבנית חוזרת. נסה לאתר את התבנית החוזרת בסדרת ההוראות הבאות:

```
twenties= amount/20;
r20=amount % 20;
tens= r20/10;
r10=r20 % 10;
fives= r10 / 5;
ones = r10 % 5;
```

שים לב שהמשתנים בצידן הימני של המשוואות מופיעים פעמיים - פעם אחת עם אופרטור החילוק /, ופעם אחת עם האופרטור %. שים גם לב שהתוצאה של כל פעולת שארית משמשת בביטוי הבא אחריה. תרשים 6.12 מדגים את התבנית החוזרת.

תרשים 6.12
תבניות חוזרות
בתוכנית התזרת העודף.



ניתן בקלות להרחיב את התוכנית על ידי החלת התבנית על רמות נוספות. לדוגמה, אם ברצונך להשתמש גם בשטרות בני חמישים דולר, יש להכניס בתוכנית את השינוי הבא:

```
fifties=amount/50;
r50=amount % 50;
twenties= r50/20;
r20=r50 % 20;
```

שתי ההוראות החדשות שמתווספות לתוכנית בנויות באותה תבנית. ההוראות המחשבות את מספר השטרות בני עשרים הדולר עוברות שינוי קטן בלבד.

איתור ותיקון תקלות

אם אינך בטוח במידת הדיוק של הפלט, נסה להוסיף לתוכנית הוראות ()printf. חלק את התהליך למספר גדול ככל האפשר של פעולות נפרדות. אחרי כל פעולה שמשנה את הערך

של משתנה, הכנס הוראת ()printf המציגה את תוכנו, גם אם אין צורך לראות את הערך בפלט הסופי של התוכנית.

הרץ את התוכנית והתבונן בפלט של אותן הוראות ()printf נוספות. השווה אותו לערכים שציפית לקבל. הפלט השגוי הראשון יכוון אותך אל הבעיה. לדוגמה, אם מוצג הפלט הבא בתוכנית חישוב השכר השבועי:

```
regular is 400
extra is -2
d_time is 40
```

ברור מייד שמשהו לקוי בערך השלילי של המשתנה extra. ניתן להניח שמיקום הבעיה הוא לפני הוראת ()printf שמציגה את המשתנה. אחרי תיקון הבעיה, אפשר למחוק את השורות המיותרות.

שאלות

1. מה ההבדל בין האופרטור / לבין האופרטור %?
2. האם ניתן לשלב טיפוסי נתונים שונים בפעולה חשבונית אחת? אם כן, כיצד ישפיע הדבר על תוצאת הפעולה?
3. מהו ביטוי?
4. היכן ניתן להשתמש בביטוי?
5. תאר את סדר הקדימויות.
6. מדוע יש להשתמש בסוגריים בפעולות חשבוניות?
7. הסבר את ההבדל בין count=count+1 לבין ++count.
8. מהו אוגר?
9. תאר אופרטור של הקצאת ערכים.
10. מה ההבדל בין count-- לבין --count?

תרגילים

1. כתוב תוכנית המחשבת מה יהיה גילו של המשתמש בשנת 2000.
2. כתוב תוכנית המחשבת את החזקה הריבועית והחזקה השלישית של מספר המוזן באמצעות המקלדת.

3. כתוב תוכנית שממירה נתוני טמפרטורה מפרנהייט לצלזיוס. נוסחת ההמרה היא celsius = (5.0/9.0)X(fahrenheit-32).

4. שנה את התוכנית שכתבת בתרגיל 3, כך שתציג את הפרש הטמפרטורה בין הערך המוקלד לבין נקודת הקיפאון, הן בצלזיוס והן בפרנהייט.

5. הסבר מה לקוי בתוכנית הבאה:

```
#define tax_rate 0.06
main()
{
    float cost, total;
    printf("Enter the cost of the item: ");
    scanf("%f", &cost);
    printf("Enter the shipping charge: ");
    scanf("%f", &shipping)'
    total = cost + cost * tax_rate + shipping;
    printf("The total is %f", total);
}
```

7

כיצד פועלות הפונקציות

אם תערום את כל חפציך בערימה אחת על הרצפה, תתקשה למצוא פריט מסוים. כדי שניתן יהיה למצוא כל חפץ בקלות, אתה ממיין את חפציך מראש ומניח אותם על-גבי מדפים. כאשר אתה זקוק לחפץ כלשהו, אתה יודע בדיוק היכן למצוא אותו.

גם תוכנית המחשב שלך עלולה לסבול מאי-סדר דומה. ככל שהתוכנית גדלה, עבודה עם פונקצית ()main גדולה אחת נעשית קשה יותר. אינך יכול להניח את קוד התוכנית על גבי מדפים, אבל אתה יכול לחלק את התוכנית לפונקציות.

פונקציה היא סדרה של הוראות שמבצעות ביחד מטלה מסוימת. באפשרותך לכתוב פונקציות משלך ולהשתמש בהן בדיוק כמו שאתה משתמש בפונקציות הכלולות בספריית פונקציות של שפת C או שפת ++C. אתה קורא לפונקציה כדי לבצע מטלה, ויכול גם להעביר ארגומנטים אל הפונקציה וממנה.

הפונקציות שימושיות במיוחד במקרים בהם עליך לבצע אותן הוראות יותר מפעם אחת. לדוגמה, אם התוכנית שלך מציגה מידע רב על-גבי המסך, יהיה עליך להשהות את התצוגה כאשר המסך מתמלא. במקום להקליד את ההוראות הגורמות להשהיית התצוגה בכל פעם שעל התוכנית לעשות זאת, באפשרותך להקליד אותן פעם אחת בלבד לתוך פונקציה נפרדת, ואז לקרוא לפונקציה בכל פעם שיש צורך להשהות את התצוגה, כמודגם בתרשים 7.1.



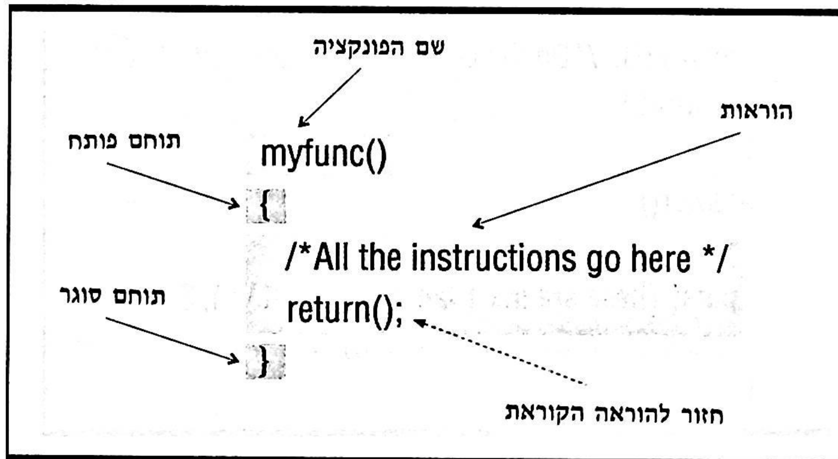
תרשים 7.1
מספר קריאות לפונקציה.

ניתן להשתמש בפונקציות גם לסייע בארגון של תוכנית קשה. נסה לחלק את התוכנית לקטעים שכל אחד מהם מבצע מטלה שלמה נפרדת. אם אתה מתקשה בכתיבת קטע כלשהו, חלק אותו שוב לקטעים קטנים יותר. המשך לחלק את התוכנית לקטעים יותר ויותר קטנים, עד שתוכל לכתוב את ההוראות המבצעות כל אחת מהמטלות. כל קטע כזה ניתן להכניס לפונקציה משלו. ברגע שכתבת את כל הפונקציות הנפרדות וקישרת ביניהן בתוך הפונקציה הראשית ()main, התוכנית מוכנה!

שימוש בפונקציות

את הפונקציות שכתבת יש להציב אחרי הסוגר המסולסל המסמן את סופה של main().
המבנה של כל אחת מהפונקציות דומה למבנה של main(), כפי שמודגם בתרשים 7.2. כמו ב-main(), אחרי שם הפונקציה מופיעים סוגריים, ללא נקודה-פסיק בסוף הביטוי. ההוראות הכלולות בפונקציה תחומות בתוך סוגריים מסולסלים.

תרשים 7.2
מבנה הפונקציה.



בעת שאתה קורא לפונקציה, המחשב מבצע את ההוראות הכלולות בפונקציה וחוזר לפקודה המופיעה מייד אחרי הקריאה לפונקציה. בקומפיילרים מסוימים אין צורך לכלול את הפקודה return(0) - C חוזרת לפקודה שאחרי הקריאה לפונקציה באופן אוטומטי, מייד עם סיום ביצוע ההוראות הכלולות בפונקציה.

הפקודה return() וקומפיילר PCC

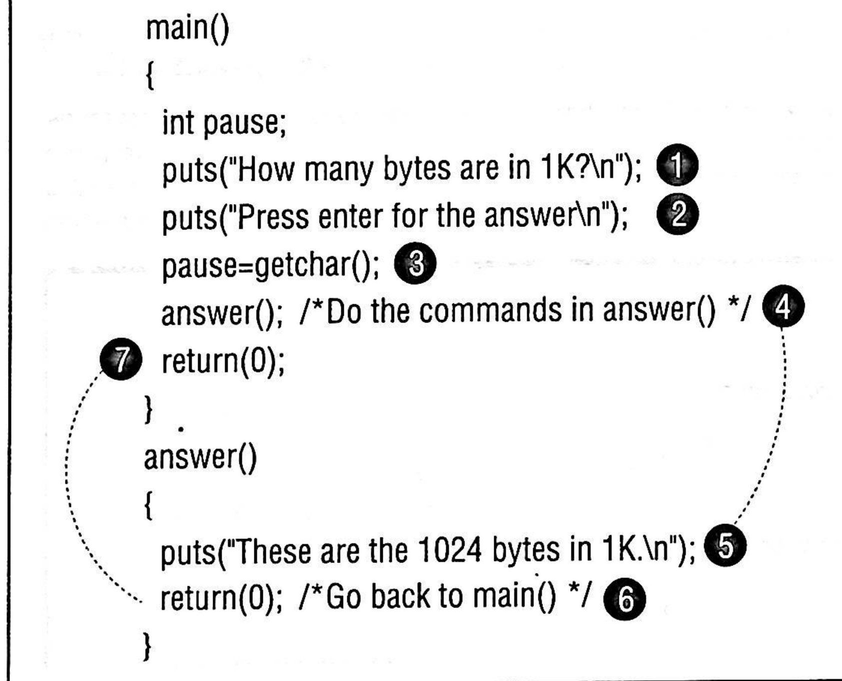
זכור, קומפיילר PCC המופיע על התקליטון המצורף לספר אינו מקבל את צורת הכתיב:

```
return;
```

במקום זאת, השתמש באחת מצורות הכתיב הבאות:

```
return();
return(0);
```

לדוגמה, תרשים 7.3 מדגים תוכנית שמציגה שאלה ותשובה. המספרים מציינים את סדר זרימת התוכנית.



תרשים 7.3
קריאה לפונקציה.

הפקודה ()answer הכלולה בתוך ()main קוראת לפונקציה, בדיוק כפי שמתבצעת קריאה לפונקציה הכלולה בספריית C. אולם, הקוד של הפונקציה ()answer נמצא בתוכנית עצמה, במקום בספרייה. אחרי שתי פקודות ()puts הראשונות בתוך ()main, הפקודה

```
answer();
```

קוראת לפונקציה. אין צורך להעביר פרמטרים ל-()answer, מכיוון שהפונקציה כוללת את כל הנתונים הדרושים לה לביצוע המטלה. בשלב זה מתבצעת הוראת ()puts שבתוך ()answer, והפקודה (return(0 מחזירה את הקומפיילר אל ההוראה המופיעה מייד אחרי הקריאה לפונקציה. הפקודה (return(0 שבתוך ()main מסיימת את התוכנית.

מבחינת המחשב, סדר ביצוע ההוראות הוא כדלקמן:

```
puts("How many bytes are in 1K?");
puts("Press Enter for the answer");
pause=getchar();
answer();
puts("There are 1024 bytes in 1K.");
return(0);
return(0);
```

ניתן לקרוא לפונקציה מכל מקום בתוכנית, אפילו מתוך פונקציה אחרת. כאשר הקומפיילר נתקל בפקודת return(0), הוא מחזיר את השליטה לשורה הבאה מייד אחרי הקריאה לפונקציה. תוכנית 7.1 כוללת שתי פונקציות שנכתבו לבד (כלומר, שאינן פונקציות ספרייה). הפונקציה הראשונה, question(), נקראת מתוך main(). הפונקציה השנייה, answer(), נקראת מתוך question(). הפקודה return(0) שבתוך answer() מחזירה את השליטה ל-question(), שמחזירה אותה לפונקציה main(). להלן הפלט של התוכנית, כשלצד כל שורה מופיעה הפונקציה שמציגה אותה:

Welcome to the Sybex Challenge.	main()
Name a graphic interface from Microsoft.	question()
Press Enter for the answer.	question()
The answer is Windows.	answer()
Thanks for your patronage.	main()

תוכנית 7.1: קריאה לשתי פונקציות

```
/*quiz2.c*/
main()
{
  puts("Welcome to the Sybex Challenge.\n");
  question();
  puts("Thanks for your patronage.\n");

  return(0);
}
question()
{
  int move_on;
  puts("Name a graphic interface from Microsoft.\n");
  puts("Press Enter for the answer."\n);
  move_on=getchar();
  answer();
  return(0);
}
answer()
{
  puts("The answer is Windows.\n");
  return(0);
}
```

השימוש הטוב ביותר בפונקציות הוא לצורך ארגון התוכנית לחלקים נפרדים. בתוך ()main מבצעים "פעולות משק-בית", כלומר - הצבת התוכנית וקריאות לפונקציות השונות. ניתן לראות זאת בתוכנית 7.2, המהווה עיבוד של תוכנית החידון. בתוכנית 7.2, ()main כוללת אך ורק קריאות לפונקציות. ה"עבודה" של התוכנית מתבצעת בפונקציות המופיעות אחרי ()main.

תוכנית 7.2: שימוש בפונקציה ()main לקריאה לכל יתר הפונקציות

```
/*quiz3.c*/
main()
{
     welcome();
     question();
     answer();
     the_end();
     return(0);
}
welcome()
{
     puts("Welcome to the Sybex Challenge.\n");
     return();
}
question()

{
     int move_on;
     puts("Name a graphic interface from Microsoft.\n");
     puts("Press Enter for the answer.\n");
     move_on=getchar();
     return(0);
}
answer()
{
  puts("The answer is Windows."\n);
  return;
}
the_end()
{
     puts("Thank you for your patronage.\n");
     return(0);
}
```

מבנה זה מקל על איתור השגיאות. לדוגמה, אם מתגלה בעיה בהצגת התשובה, חפש את השגיאה בפונקציה ()answer.

הפונקציות מבוצעות על-פי הסדר בו הן נקראות, ולא על-פי סדר הופעתן בתוכנית. יכולנו להציב את הפונקציות בתוכנית 7.2 גם בסדר הבא:

```
main()
answer()
welcome()
the_end()
question()
```

והפלט לא היה משתנה, מכיוון שהפונקציות נקראות בסדר הנכון מתוך ()main.

משתנים בפונקציות

כאשר התוכנית כוללת פונקציות נוספות, מלבד ()main, יש לשקול היכן וכיצד להכריז על המשתנים. C כוללת מספר טיפוסי משתנים. בפרק זה נבחן את המשתנים האוטומטיים, החיצוניים והסטטיים.

משתנים אוטומטיים (מקומיים)

פונקציות מסוימות זקוקות למשתנים או קבועים משלהן. לדוגמה, הבה נבחן את הבעיה של השהיית התצוגה על המסך אחרי הצגת סדרה ארוכה של הודעות. ניתן להשהות את התצוגה בעזרת ההוראות הבאות:

```
printf("Press any key to continue");
move_on = getchar();
```

ובמקרה שעליך להשהות את התצוגה מספר פעמים במהלך התוכנית, הצב את ההוראות האלה בתוך פונקציה נפרדת. לאחר מכן, בכל פעם שיש להשהות את התצוגה, כל שעליך לעשות הוא לקרוא לפונקציה. מכיוון ש-()getchar זקוקה למשתנה, עליך להכריז על משתנה בתוך הפונקציה, כך:

```
pause()
{
      int move_on;
      printf("Press any key to continue");
      move_on = getchar();
return(0);
}
```

המשתנה move_on הוא חלק מהפונקציה ()pause. מכיוון שהוא מוכרז בתוך פונקציה, הוא מכונה משתנה *מקומי* לאותה פונקציה. משתנה מקומי מוכר רק על ידי הפונקציה בתוכה הוא מוכרז. בשפת C נקרא משתנה כזה *אוטומטי*.

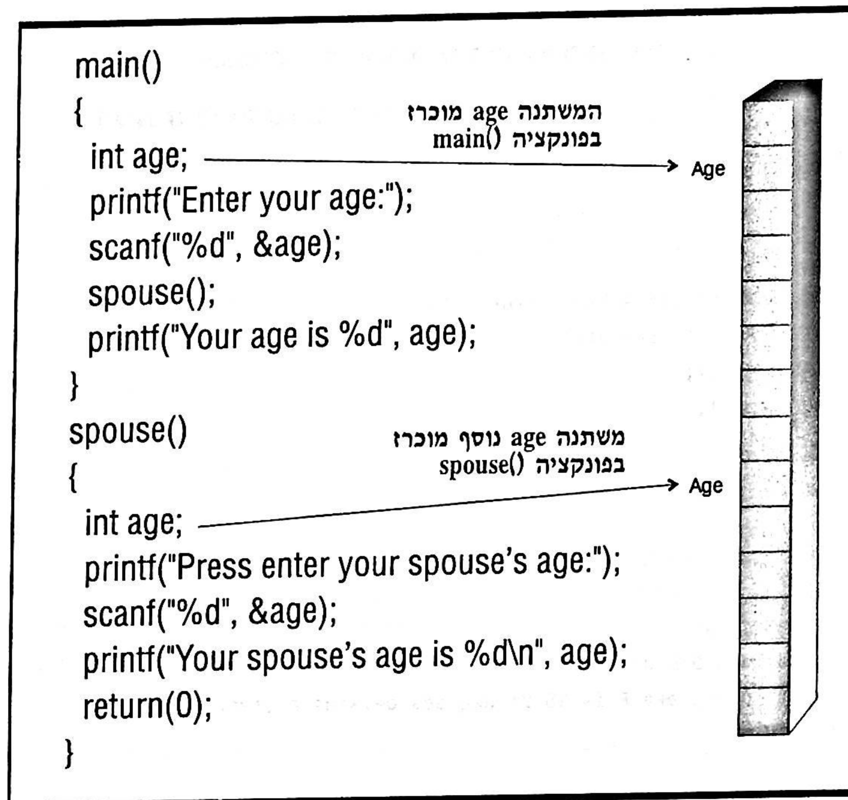
חשוב על כך. אם אתה מכריז על משתנה בתוך פונקציה, הוא מוכר רק במסגרת אותה פונקציה. הדבר נכון גם לגבי משתנים שמוכרזים בתוך הפונקציה ()main. עיין בתוכנית הבאה:

```
main()
{
     int move_on;
     puts("Messages that fill the screen\n");
                   /* Here would be a series of messages
                      that fill the screen */
     pause();
}
pause()
{
     printf("Press any key to continue");
     move_on = getchar();
return(0);
}
```

הקומפיילר יציג הודעת שגיאה מכיוון שלא ניתן להשתמש במשתנה move_on בפונקציה ()pause. המשתנה הוכרז בפונקציה ()main. לפיכך, הוא נחשב משתנה מקומי לפונקציה ()main, והוא בעל משמעות אך ורק כאשר עושים בו שימוש בתוך ()main. כדי לפתור את הבעיה, עליך להכריז על המשתנה move_on בתוך הפונקציה ()pause.

מכיוון שמשתנה אוטומטי תקף רק בתוך הפונקציה בה הוכרז, ניתן להעניק שם זהה למשתנים שמוכרזים בפונקציות שונות. בתרשים 7.4, המשתנה ששמו age מוכרז הן בפונקציה ()main והן בפונקציה ()spouse. כל אחד מהם הוא משתנה אוטומטי, או מקומי, לגבי הפונקציה שבתוכה הוא מוכרז. כלומר, בתוכנית זו ישנם שני משתנים שונים בעלי שם זהה. כל אחד מהם יכיל ערך משלו שיאוחסן בעמדת זיכרון נפרדת. המחשב מבחין ביניהם ויודע איזה משתנה age ברצונך להציג לפי הפונקציה בה כלולה הוראת ה-()printf. כאשר המחשב מבצע את הוראת ()printf שבפונקציה ()spouse, הוא מציג את המשתנה age האוטומטי לפונקציה זו. כאשר המחשב מבצע את ההוראה ()printf שבפונקציה ()main, הוא מציג את המשתנה age האוטומטי לפונקציה ()main. המחשב שומר בזיכרון שני משתנים נפרדים ששמם age.

תרשים 7.4
משתנים מקומיים יכולים להיות בעלי שם זהה בפונקציות שונות.



משתנים חיצוניים (גלובליים)

משתנה חיצוני הוא משתנה שניתן להשתמש בו בכל פונקציה בתוכנית. בשפות תכנות אחדות מכונים משתנים כאלה בשם משתנים גלובליים. כדי שמשתנה ישמש כמשתנה חיצוני, יש להכריז עליו לפני הפונקציה ()main, כך:

```
int temp;
main()
```

בדוגמה זו, המשתנה temp הוא משתנה חיצוני. לפיכך, כל אחת מהפונקציות בתוכנית יכולה לעשות בו שימוש. עיין בתוכנית 7.3. מכיוון ש-temp הוא משתנה חיצוני, ניתן להשתמש בו בפונקציות ()main, ()convert, ()freeze ו-()boil. הערך מוזן למשתנה בפונקציה ()main, ויתר הפונקציות משתמשות בו לצורך ביצוע החישובים הדרושים. התוכנית עושה שימוש במשתנה נוסף - celsius. מכיוון שהפונקציה היחידה שזקוקה

למשתנה זה היא ()convert, הכרזת המשתנה מתבצעת בתוכה. זהו משתנה מקומי לפונקציה ()convert, ולא ניתן להשתמש בו במקום אחר בתוכנית.

תוכנית 7.3: שימוש במשתנה חיצוני

```
/*f_to_c.c*/
int temp;
main()
{
     printf("Input a temperature :");
     scanf("%d", &temp);
     convert();
     freeze();
     boil();
}

convert()
{
float celsius;
celsius = (5.0/9.0)*(temp-32);
printf("%d degrees F is %6.2f degrees celsius\n",temp, celsius);
return(0);
}
freeze()
{
printf("It is %d degrees from the freezing point\n",temp-32);
return(0);
}
boil()
{
printf("It is %d degrees from the boiling point\n", 212-temp);
return(0);
}
```

אם התוכנית כוללת משתנה חיצוני ומשתנה מקומי בעלי שם זהה, הפונקציה תשתמש במשתנה המקומי. היא אינה יכולה להשתמש בשניהם.

משתנים סטטיים

משתנה אוטומטי (מקומי) מתקיים רק בשעה שהפונקציה בתוכה הוא מוכרז מתבצעת. כאשר הפונקציה מתחילה להתבצע, מוקצית למשתנה עמדת זיכרון. עם גמר ביצוע הפונקציה, משתחררת עמדת הזיכרון והמשתנה חדל להתקיים.

אם הפונקציה מבוצעת יותר מפעם אחת, המשתנה המקומי נוצר מחדש בכל פעם שהפונקציה מתחילה להתבצע, ובכל פעם עשויה להיות לו כתובת זיכרון שונה. הערך המאוחסן במשתנה אובד בכל פעם שביצוע הפונקציה מסתיים.

באפשרותך לשמור על הערך של המשתנה המקומי גם כאשר הפונקציה מסתיימת. לשם כך עליך להכריז על המשתנה כמשתנה סטטי. ההכרזה מבוצעת כך:

```
myfunc()
{
      static int count;
}
```

למשתנה סטטי מוקצית עמדת זיכרון קבועה, למשך כל זמן הרצת התוכנית. כאשר מסתיים ביצוע הפונקציה, עמדת הזיכרון אינה משתחררת אלא ממשיכה לאחסן את הערך שהיה למשתנה בעת שהפונקציה הסתיימה. זה יהי ערכו של המשתנה בפעם הבאה שהפונקציה תבוצע. זכור שמדובר עדיין על ערך אוטומטי (מקומי), שניתן להשתמש בו רק בפונקציה בתוכה הוכרז.

עיין בתוכנית 7.4, שאינה משתמשת במשתנה סטטי. הפונקציה ()doit מבוצעת ארבע פעמים. בכל פעם שישנה קריאה לפונקציה, מקבלים שני המשתנים את הערך 0, והערכים מוצגים בהוראת ()printf. עם סיום הפונקציה מתווסף הערך המילולי 1 לכל אחד מהמשתנים. עם זאת, המשתנים משתחררים אחרי כל ביצוע של הפונקציה ומקבלים מחדש את הערך 0 בפעם הבאה שהפונקציה מבוצעת. הפלט הוא לפיכך כדלקמן:

```
autovar is 0   statvar is 0
autovar is 0   statvar is 0
autovar is 0   statvar is 0
autovar is 0   statvar is 0
```

תוכנית 7.4: הדגמת השימוש במשתנים מקומיים

```
/*stat.c*/
main()
{
      doit();
      doit();
      doit();
      doit();
```

```
}
doit()
{
     int autovar = 0;
     int statvar = 0;
     printf("autovar is %d  statvar is %d\n", autovar, statvar);
     ++autovar;
     ++statvar;
return(0);
}
```

כעת, נשנה את ההכרזה בפונקציה ()doit וניצור משתנה סטטי:

```
static int statvar = 0;
```

כאשר מריצים את התוכנית עם משתנה סטטי, הפלט המתקבל הוא:

```
autovar is 0   statvar is 0
autovar is 0   statvar is 1
autovar is 0   statvar is 2
autovar is 0   statvar is 3
```

בפעם הראשונה שהתוכנית מורצת, מוקצה למשתנה statvar הערך ההתחלתי 0. ההכרזה על המשתנה כמשתנה סטטי, עם זאת, מורה לתוכנית לשמור בזיכרון את הערך המאוחסן במשתנה בעת שהפונקציה מסתיימת. עם סיום הפונקציה בפעם הראשונה, ערך זה הוא 1 - מכיוון שהשתמשנו באופרטור ++statvar. כאשר הפונקציה מבוצעת בפעם השנייה, ערכו של המשתנה עדיין 1; התוכנית מתעלמת מהערך ההתחלתי 0.

בכל פעם שהפונקציה מתבצעת, ערכו של המשתנה גדל ב-1. הערך החדש נשמר בזיכרון ומשמש כערך התחלתי כאשר הפונקציה מבוצעת פעם נוספת.

העברת פרמטרים

ישנן מטלות מסוימות שניתן לבצען רק על ידי העברת פרמטר לפונקציה. כאשר מעבירים פרמטר לפונקציה ()puts, למשל, יש להקליד את הארגומנט - המחרוזת שאותה רוצים להציג - בתוך הסוגריים. הקריאה לפונקציה ()puts מתבצעת בעזרת פקודה כזו:

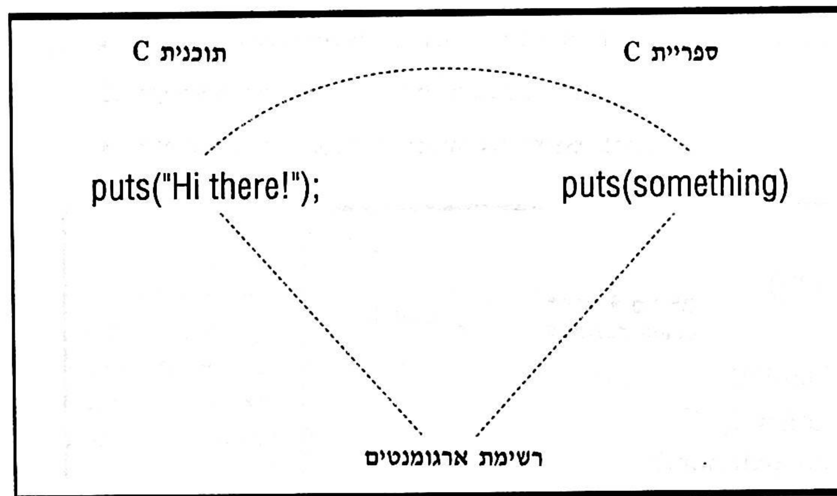
```
puts("Hi there!");
```

המחרוזת "!Hi there" משוגרת אל פונקצית הספרייה ()puts ומורה לה מהו המידע שברצונך להציג על המסך.

העברת פרמטרים אל פונקציות שכתבת בעצמך מתבצעת באופן דומה. עליך להקליד בתוך הסוגריים את כל הנתונים שברצונך לשגר אל הפונקציה. רשימת המשתנים המועברת לפונקציה נקראת *רשימת ארגומנטים*.

הבה נבחן עתה מה קורה בצד המקבל. פונקצית הספרייה ()puts כוללת הוראה האומרת "הציגי משהו על-גבי המסך". הפונקציה מצפה שיועבר אליה אותו "משהו" - פרמטר. פונקצית הספרייה זקוקה למקום כלשהו כדי לאחסן את המחרוזת המועברת אליה, כפי שמודגם בתרשים 7.5. לפיכך, גם לה יש רשימת ארגומנטים המכילה את שמות המשתנים שיאחסנו את הנתונים המתקבלים. ניתן להעביר יותר מארגומנט אחד, בתנאי שמספר הארגומנטים בפקודת הקריאה לפונקציה, וכן טיפוסי הנתונים של אותם ארגומנטים - מתאימים למספר ולטיפוסי הנתונים שהפונקציה מצפה לקבל.

תרשים 7.5
יש צורך בארגומנט כדי לקבל ערך מועבר.



הבה נראה כיצד להעביר ארגומנט לפונקציה שכתבת בעצמך. עיין בתוכנית הבאה:

```
main()
{
      int count;
      count = 5;
      doubles(count);
}
doubles(num)
int num;
```

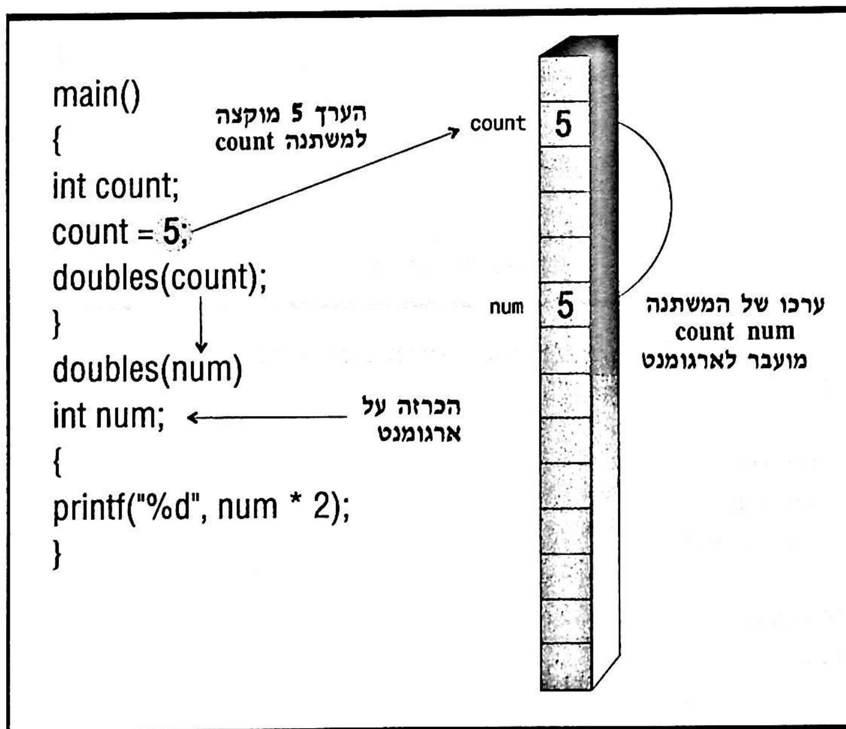
```
{
    printf("%d", num * 2);
return(0);
}
```

הפקודה ;doubles(count) בפונקציה ()main קוראת לפונקציה ומעבירה אליה את ערכו של המשתנה count. הפונקציה ()doubles מקבלת את הערך של הארגומנט ומאחסנת אותו במשתנה ששמו num. הערך של המשתנה num יהיה, לפיכך, זהה לערך של המשתנה count. שים לב שהמשתנה num מוכרז לפני הסוגר המסולסל הפותח את הפונקציה ()doubles. שורה זו נקראת הכרזת הארגומנט והיא מורה לקומפיילר איזה טיפוס נתונים יתקבל על ידי הפונקציה. רק הכרזות ארגומנטים יכולות להופיע לפני התוחם הפותח את הפונקציה. אם הפונקציה זקוקה למשתנים נוספים, יש להכריז עליהם אחרי התוחם הפותח, באופן הרגיל.

להלן שלבי התהליך (עיין גם בתרשים 7.6):

1. מתבצעת קריאה לפונקציה ()doubles ומועבר אליה הערך של המשתנה count.
2. הערך 5 מתקבל בפונקציה על ידי הארגומנט ששמו num.
3. הפונקציה מכפילה את הערך ומציגה את התוצאה בהוראת ()printf.

תרשים 7.6
העברת פרמטר.



הערות ++C

בשפת ++C ניתן להכריז על הארגומנטים ישירות ברשימת הארגומנטים של הפונקציה, כך:

```
doubles(int num)
```

כפי שניתן לראות בתוכנית 7.5, באפשרותך להעביר אל הפונקציה מספר פריטי נתונים, מכל טיפוס נתונים שהוא, בהתאם לצורך. הפונקציה ()area מחשבת את שטחה של קומה. נתוני האורך והרוחב של הקומה, וכן מספרה, מוזנים בפונקציה ()main, ואז מועברים אל הפונקציה בפקודת הקריאה:

```
area(length, width, fnum);
```

תוכנית 7.5: העברת כמה פרמטרים

```
/*area.c*/
main()
{
     float length, width;
     int fnum;
     printf("Enter the floor number: ");
     scanf("%d", &fnum);
     printf("Enter the length of the floor: ");
     scanf("%f", &length);
     printf("Enter the width of the floor: ");
     scanf("%f", &width);
     area(length, width, fnum);
}
area(size, wide, num)
float size, wide;
int num;
{
     float area;
     area=size*wide;
     printf("The area of floor %d is %.2f", num, area);
return(0);
}
```

שלושת הארגומנטים נקלטים בסדר בו הם מועברים. במקרה זה, הערך של length נקלט על ידי size, הערך של width נקלט על ידי wide, והערך של fnum נקלט על ידי num. טיפוסי הנתונים של הארגומנטים מתאימים - שני ערכים מטיפוס float נקלטים על ידי משתנים מטיפוס float, וערך מטיפוס int נקלט על ידי משתנה מטיפוס int.

הפונקציה ()area זקוקה גם למשתנה שיאחסן את תוצאת החישוב. משתנה זה, הנקרא אף הוא area, מוכרז אחרי התוחם הפותח.

אם שגית וכתבת את הקריאה לפונקציה כך:

```
area(width,length,fnum);
```

הערך של width ייקלט על ידי size, והערך של length ייקלט על ידי wide. מכיוון שטיפוסי הנתונים מתאימים, הקומפיילר לא יציג הודעת שגיאה, ובמקרה זה התוכנית עדיין תגיע לתוצאה הנכונה - סדר האיברים במכפלה אינו משנה. לעומת זאת, נניח שכתבת את רשימת הארגומנטים בקריאה לפונקציה כך:

```
area(fnum, width,length);
```

הפונקציה תקלוט את fnum לתוך הארגומנט size, את width לתוך הארגומנט wide, ואת length לתוך num. שני ארגומנטים אינם מתאימים מבחינת טיפוס הנתונים. גם אם הקומפיילר שברשותך אינו מציג הודעת שגיאה, תוצאת החישוב תהיה ללא ספק שגויה.

נבחן דוגמה נוספת, בתוכנית 7.6. התוכנית מזינה את מחירו של פריט ואחוז הנחה. המשתנים cost ו-discount מועברים אל הפונקציה ()price ונקלטים לתוך הארגומנטים amount ו-mrkdown. המשתנים reduced ו-net נדרשים בתוך הפונקציה, ולכן הם מוכרזים כמשתנים אוטומטיים אחרי התוחם הפותח.

תוכנית 7.6: העברת פרמטרים

```
/*discount.c*/
main()
{
float cost, discount;
printf("Enter the cost of the item: ");
scanf("%f", &cost);
printf("Enter the percent discount as decimal number: ");
scanf("%f", &discount);
price(cost, discount);
}
price(amount, mrkdown)
float amount, mrkdown;
{
float reduced, net;
```

```
reduced = amount * mrkdown;
net = amount - reduced;
printf("The net cost is $%.2f", net);
return(0);
}
```

הפונקציה ()price מכפילה את ערך משתנה המחיר (cost) בערכו של משתנה ההנחה (discount), מפחיתה את ההנחה מהמחיר, ומציגה את המחיר אחרי ההנחה. תצוגת המסך אחרי הרצת התוכנית עשויה להיראות כך:

```
Enter the cost of the item: 100
Enter the percent discount as decimal number: 0.05
The net cost is $95.00
```

עתה, נניח שטעית וכתבת את הקריאה לפונקציה כך:

```
price(discount, cost);
```

הקומפיילר לא יציג הודעת שגיאה, מכיוון ששני ערכים מטיפוס float נקלטים על ידי שני ארגומנטים מטיפוס float. אך לרוע המזל, הם אינם נקלטים על ידי הארגומנטים הנכונים. הערך של discount נקלט על ידי amount, ואילו הערך של cost נקלט על ידי mrkdown.

אם המשתמש מקליד את הערך 100 עבור המחיר, ו-0.05 עבור ההנחה, הפונקציה תהפוך את הסדר. היא תחשב את מחיר הפריט כחמישה סנט ואת ההנחה כ-100%. מחיר הפריט אחרי ההנחה יהיה מינוס $4.95 (-4.95) במקום $95.00.

החזרת משתנים

פונקציה יכולה לא רק לקלוט ערכים, אלא גם להחזירם. זכור לך בוודאי שהפונקציה ()getchar משתמשת בפורמט הבא:

```
key = getchar();
```

ההוראה קוראת לפונקציה ()getchar, שמזינה תו אחד מהמקלדת. בדוגמה שלפנינו, כאשר מבוצעת הפונקציה ()getchar, תו הקלט מוצב במשתנה ששמו key. פעולה זו נקראת *החזרת ערך*.

הפונקציות שלך יכולות להחזיר ערכים לפונקציה הקוראת. עם זאת, כדי להחזיר ערכים, יש להוסיף לפונקציה כמה מרכיבים. לדוגמה, עיין בתוכנית הבאה:

```
main()
{
    char letter;
    letter = getlet();
```

```
        putchar('\n');
        printf("The character entered was %c", letter);
}
char getlet()
{
        printf("Enter a character: ");
        return(getchar());
}
```

ההוראה:

```
letter = getlet();
```

קוראת לפונקציה ()getlet. פונקציה זו מזינה תו מהמקלדת ומחזירה את התו למשתנה letter בפונקציה ()main.

כאשר אתה רוצה להחזיר ערך, עליך למסור לקומפיילר מהו טיפוס הנתונים של הערך שיוחזר. הדבר מתבצע על ידי ציון טיפוס הנתונים לפני שמה של הפונקציה, כמו בדוגמה שלנו:

```
char getlet()
```

הוראה זו אומרת ל-C שהפונקציה ()getlet תחזיר ערך מטיפוס char. הערך שיוחזר הוא הארגומנט המופיע בקריאה לפונקציה ()return. ההוראה:

```
return(getchar());
```

מבצעת את עיקר העבודה. זכור שהפונקציה ()getchar משמשת במקום בו היינו מציבים ביטוי או ערך. בתוכנית שלנו, ()getchar מזינה תו, ()return מעבירה את התו בחזרה למשתנה ששמו letter ומחזירה את השליטה לפונקציה ()main, בתוכה מתבצעת הוראת ()printf.

ערכים מוחזרים מסוג "Void"

בעבודה עם קומפיילרים של שפת C הפועלים לפי תקן ANSI, עליך לקבוע את טיפוס הפונקציה, גם כאשר הפונקציה אינה מחזירה ערך כלשהו. במקרים כאלה, טיפוס הנתונים מוכרז כ-"void", כמו בדוגמה להלן:

```
Void Myfunct()
```

הכרזה זו מורה לקומפיילר שהפונקציה אינה מחזירה ערך לפונקציה הקוראת. בשפת ++C אין הכרח להשתמש במלת המפתח void, אם כי מומלץ לעשות זאת.

בחן היטב את אופן השימוש בפונקציה ()return בתוכנית זו. ודא שאתה מבין כיצד פועלות פונקציה זו והפונקציה ()getchar.

נבחן עתה תוכנית שקולטת ערך ומחזירה אותו. התוכנית המודגמת בתרשים 7.7 מזינה ערך ומציגה את חזקתו הריבועית. ההוראה:

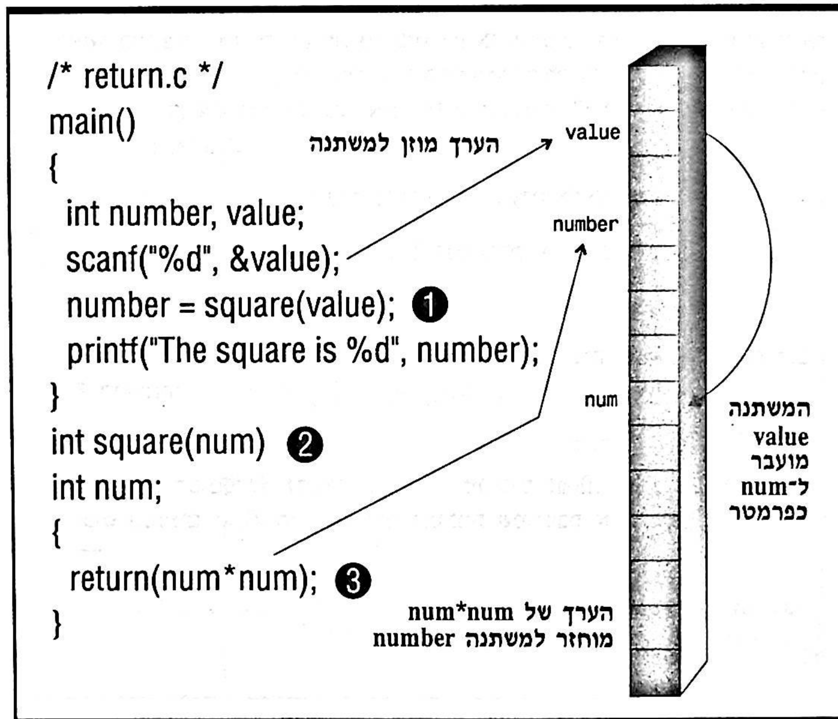
```
number = square(value);
```

קוראת לפונקציה square() ומעבירה אליה את תוכנו של המשתנה value.

הגדרת הפונקציה int square(num) מוסרת ל-C שהפונקציה square() תחזיר ערך מטיפוס int, ותקלוט ארגומנט לתוך המשתנה num.

בדוגמה שלנו, return() היא ההוראה היחידה הכלולה בפונקציה. ההוראה return(num*num) מחזירה את הערך המחושב של הביטוי num*num ומחדירה אותו למשתנה ששמו number - המשתנה שהוקצה לקריאה לפונקציה בתוך main().

תרשים 7.7
תוכנית שמחזירה ערך מפונקציה.



הערות ++C

בעבודה עם קומפיילרים של ++C (וכן עם קומפיילרים מסוימים של C), מומלץ לפתוח את התוכנית עם אב-טיפוס של פונקציה. אב-טיפוס הוא שורת ההכרזה של הפונקציה, הנכתבת בתחילת התוכנית, לפני ()main, כך:

```
void square(int num);
main()
```

אב-הטיפוס מוסר לקומפיילר באיזה טיפוס פונקציה ובאילו ארגומנטים תעשה שימוש בתוכנית.

החזרת ערכים מטיפוס float

כאשר פונקציה מחזירה משתנה מטיפוס int או מטיפוס char, אין הכרח לציין את טיפוס הנתונים לפני שמה של הפונקציה. C תוכננה מלכתחילה לטפל בנתונים מטיפוס int ו-char בלבד. לפיכך, אם אין מציינים את טיפוס הנתונים, C מניחה שבכוונתך להחזיר ערך מטיפוס int או char.

לעומת זאת, אם אתה מחזיר ערך מטיפוס float, עליך לבצע שני דברים:

1. לכתוב את טיפוס הנתונים float לפני שמה של הפונקציה.
2. להכריז על הפונקציה עצמה.

מכריזים על הפונקציה כעל משתנה חיצוני לפני הפונקציה ()main, כפי שמודגם בתרשים 7.8. ההוראה:

```
float square();
```

מכינה את הקומפיילר לטיפול בפונקציה מטיפוס float. מלבד הכרזת הפונקציה וציון טיפוס הנתונים float, התוכנית זהה לתוכנית שמחשבת את חזקתו הריבועית של מספר שלם.

מרבית הקומפיילרים מאפשרים גם להכריז על טיפוס הפונקציה בתוך ()main, כך:

```
main()
{
float number, value, square();
```

הערות ++C

העמסת יתר היא תהליך בשפת ++C המאפשר לאופרטורים ולפונקציות לטפל ביותר מטיפוס נתונים אחד. ניתן להעניק שם זהה למספר פונקציות, כפי שמודגם בשורות אב-הטיפוס הבאות:

```
int doubles(int num);
float doubles(float num);
```

התוכנית תכלול שתי פונקציות ששמן ()doubles. פונקציה אחת משמשת להכפלת מספר שלם, והשנייה להכפלת מספר מטיפוס float.

תרשים 7.8
הכרזה על פונקציה מטיפוס float.

```
/* float.c */
float square();
main()
{
 float number, value;
 printf("Enter a number to be squared:");
 scanf("%d", &value):
 number = square(value);
 printf("The square is %.2f", number);
}
float square(num)
float num;
{
 return(num*num);
}
```


אם הקומפיילר שברשותך אינו מאפשר זאת, הכרז על הפונקציה לפני ()main.

הפונקציה ()return ב-()main

לפני שנמשיך, הבה ניזכר במבנה של הפונקציה return(0) ב-()main. אם ראינו שאנו משתמשים בפרמטר שבסוגריים כדי לשגר ערך כלשהו אל פונקציה, נשאלת השאלה לאן משוגר הערך 0 כאשר מסיימים את התוכנית? למעשה, אָנו משגרים אותו אל מערכת ההפעלה.

בעת שמריצים תוכנית C, ניתן לחשוב על מערכת ההפעלה כמבצעת *קריאה* לפונקציה ()main. עם סיום התוכנית, הפקודה ()return מחזירה את השליטה למערכת ההפעלה. הפרמטר המופיע בסוגריים ב-()return מורה למערכת ההפעלה אם התגלתה שגיאה, ובאיזה סוג של שגיאה מדובר. הפקודה return(0) מורה למערכת ההפעלה שלא התגלתה שגיאה. תוכניות מתקדמות יכולות לעשות שימוש בפרמטרים אחרים כדי לדווח למערכת ההפעלה על שגיאות חומרה או תוכנה, וכן להורות לה לבצע פעולות נוספות בהתאם לאופן בו הסתיימה הרצת התוכנית.

שימוש בהגדרות מקרו

למדת כבר שבאפשרותך ליצור קבוע בעזרת ההנחיה define#. לדוגמה, הפקודה #define PI 3.14 מורה ל-C להציב את הערך 3.14 בכל מקום שמופיע בתוכנית שם הקבוע PI.

ההנחיה define# מורה לקומפיילר להחליף את שמו של הקבוע בנתון המופיע אחריו בהנחיה. גם אם תופיע הוראה מלאה אחרי שם הקבוע בהנחיית define#, הקומפיילר יבצע את ההחלפה. לדוגמה, ההנחיה הבאה מחליפה את הקבוע ששמו ENTER בהוראת ()printf:

```
#define ENTER printf("Please enter a number:")
```

כאשר ברצונך להציג את ההודעה הכלולה בפקודה ()printf, כל שעליך לעשות הוא להשתמש בהוראה ENTER, כך:

```
#define ENTER printf("Please enter a number:")
main()
{
    int age, size;
    ENTER;
    scanf("%d", &age);
    ENTER;
    scanf("%d", &size);
}
```

כאשר תריץ את התוכנית, תוצג ההודעה :Please enter a number, בדיוק כאילו שהוראת ה-()printf הופיעה בתוך ()main, כפי שמודגם בתרשים 7.9.

תרשים 7.9
הגדרה של הוראה מלאה.

```
#define ENTER  printf("Please enter a number:")
main()
{
int age, size
ENTER;
scanf("%d", &age);
ENTER;
scanf("%d", &size);
}
```

ההוראה ()printf שמוקצית למקרו ENTER מחליפה את ENTER בתוכנית.

לכן, התוכנית נראית לקומפיילר כך.

```
#define ENTER  printf("Please enter a number:")
main()
{
int age, size
printf("Please enter a number:");
scanf("%d", &age);
printf("Please enter a number:");
scanf("%d", &size);
}
```

הנחיות מסוג זה נקראות *פקודות מקרו*. הן משמשות כלי-עזר שימושי ורב-עוצמה, מכיוון שהן חוסכות את הצורך לחזור ולהקליד הוראות שחוזרות מספר פעמים באותה תוכנית. באפשרותך להשתמש במקרו ENTER, למשל, בכל פעם שברצונך להנחות את המשתמש להקליד נתון כלשהו. פקודות מקרו הן שימושיות במיוחד מכיוון שביכולתן לקלוט ארגומנטים, בדומה לפונקציות. התוכנית הבאה, לדוגמה, כוללת מקרו המשמש להמרת נתוני טמפרטורה מפרנהייט לצלזיוס:

```
#define ENTER printf("Please enetr a number: ");
#define CONVERT(temp) (5.0/9.0)*(temp-32)
main()
```

```
{
    float climate;
    ENTER;
    scanf("%f", &climate);
    printf("it is %f\n", CONVERT(climate));
}
```

הנחיית #define השנייה יוצרת מקרו ששמו CONVERT המקבל ערך יחיד כארגומנט. הארגומנט ש-CONVERT מקבל מוצב בביטוי (5.0/9.0)*(temp-32) במיקום של המלה temp. הקריאה למקרו דומה לקריאה לפונקציה, תוך שימוש בערך שיש להמיר כארגומנט הקריאה (ראה תרשים 7.10). אם תקליד את הערך 212 בתגובה לפקודת הקלט, תחושב הקריאה CONVERT(climate) הכלולה בפקודה ()printf כך: (5.0/9.0)*(212-32).

תרשים 7.10
הגדרת ביטוי כמקרו.

```
#define CONVERT(temp) (5.0/9.0)*(temp-32)
```

הקומפיילר מציב את הביטוי
במקום המקרו ששמו CONVERT
ומעביר אליו את הארגומנט.

```
    scanf("%f", &climate);
    printf("It is %f\n", CONVERT(climate));

printf("It is %f\n", (5.0/9.0)*(climate-32));
```

המחשב מעריך את הביטוי
באמצעות הערך שהועבר אליו.

```
float convert(temp)
float temp
{
  return(5.0/9.0)*(temp-32);
}
```

המקרו מבוצע בדיוק כאילו היה
פונקציה שקולטת ומחזירה ארגומנט.

השימוש במקרו מעניק את כל היתרונות של השימוש בקבוע. אם ברצונך לשנות את הנחיית הקלט הרגילה שלך, למשל, כל שעליך לעשות הוא לשנות את המקרו ENTER בתחילת

התוכנית. כך, במקרה ששגית בהקלדת נוסחת ההמרה - יהיה עליך לתקן שורה אחת בלבד, במקום לתקן את הנוסחה בכל מקום בתוכנית בו השתמשת במקרו.

עיצוב התוכנית

אין כללים מוחלטים המסייעים למתכנת להחליט מתי להשתמש בפונקציות ומתי לכלול את כל ההוראות בתוך ()main. כשתצבור ניסיון בתכנות, תגלה שההחלטה מתקבלת באופן טבעי. אם אתה כותב תוכנית קצרה, כמו תוכנית שמזינה מספר אחד או שניים, מבצעת כמה פעולות מתמטיות ומציגה את התוצאות - השתמש ב-()main בלבד. אין צורך לחלק תוכנית בת חמש או עשר שורות לפונקציות נפרדות, אלא אם כן הדבר נדרש.

לעומת זאת, אם אתה כותב תוכנית ארוכה יותר בתוך ()main ומשהו משתבש, קשה הרבה יותר לאתר את הבעיה. אם התוכנית מחולקת לפונקציות נפרדות, ניתן לאתר את הבעיה באופן שיטתי, על ידי בדיקת הפונקציות אחת לאחת, החל מהפונקציה שהסבירות שהיא מקור הבעיה היא הגדולה ביותר, ועד לפונקציה בעלת הסבירות הנמוכה ביותר.

משתנים אוטומטיים לעומת משתנים חיצוניים

מתכנתים מתחילים בוחרים, לעתים קרובות, להשתמש בקיצור-הדרך של הכרזת משתנים חיצוניים בלבד, מתוך תקווה שכך יחסכו את הטרחה הכרוכה בהעברה ובהחזרה של ארגומנטים. לדוגמה, תוכנית 7.3 בתחילת הפרק עושה שימוש במשתנה החיצוני temp בארבע פונקציות שונות. מכיוון שזהו משתנה חיצוני, לא היה צורך להעביר את ערכו כארגומנט.

אם כי נראה שקל יותר להשתמש במשתנים חיצוניים, הם עלולים לגרום לשגיאות הרצה שקשה מאד לאתרן - וכתוצאה מכך, לתוצאות בלתי מדויקות. יש לזכור שאמנם כל הפונקציות יכולות להשתמש במשתנה חיצוני, אך הן יכולות גם לשנות אותו. אם קיבלת תוצאות שגויות, יהיה עליך לחפש את השגיאה בתוכנית כולה. ייתכן שהוראה קטנה ותמימה אחת, הקבורה עמוק בתוך פונקציה שנקראת רק לעתים רחוקות, שינתה את ערכו של המשתנה בטעות.

השימוש במשתנים אוטומטיים ובהעברת ערכים בארגומנטים מעניק לך שליטה רבה יותר בזרימת התוכנית. ערכו של משתנה אוטומטי יכול להשתנות רק בתוך הפונקציה בה הוא מוכרז. אם התוצאות שגויות, באפשרותך להוסיף הוראות ()printf זמניות לכל פונקציה ולהציג את הערכים של המשתנים המקומיים. כך תוכל לאתר בקלות את ההוראות השגויות.

קלט בלתי תקף

התוכניות שהוצגו בפרק זה נבחרו כדי להדגים את מושג השימוש בפונקציות. הן אינן מייצגות בהכרח את הדרך הטובה ביותר לביצוע מטלה, והן אינן מובטחות מפני שגיאות.

לדוגמה, גם התוכנית הטובה ביותר אינה יכולה לשלוט בקלט המוזן באמצעות המקלדת. בתוכנית 7.6, למשל, מחיר הפריט ושיעור ההנחה מוזנים באמצעות הפונקציה ()scanf. חישוב המחיר אחרי הנחה מבוסס על ההשערה ששיעור ההנחה יוקלד כמספר עשרוני, כדוגמת 0.05 עבור חמישה אחוזים. כדי להגדיל את הסיכויים שיוקלד קלט מתאים, מציגה הוראת ה()printf את ההנחיה:

Enter the percent discount as decimal number:

אולם, המשתמש עלול עדיין לשגות ולהקליד 5 במקום 0.05. במקרה כזה, במקום להפחית את עלות הפריט בחמישה אחוזים, שיעור ההנחה יחושב כ-500 אחוז. כלומר, המוכר משלם לקונים כדי שיקנו ממנו!

גם בתוכניות שאתה כותב, אל תניח מראש שהקלט יהיה כפי שאתה מצפה. להיפך, עליך ללמוד לצפות לבלתי-צפוי. בפרק 8 תלמד כיצד למנוע חלק מהשגיאות האלה.

שאלות

1. מה ההבדל בין פונקציה בספריית C לבין פונקציה שאתה כותב בעצמך?
2. האם פונקציה חייבת להיקרא תמיד מתוך ()main?
3. מה קורה אחרי שפונקציה מתבצעת?
4. הסבר את ההבדל בין משתנים אוטומטיים למשתנים חיצוניים.
5. מהם היתרונות היחסיים של השימוש במשתנים אוטומטיים, ומהם יתרונותיהם של משתנים חיצוניים?
6. מתי תשתמש במשתנה סטטי?
7. כיצד מעבירים ערך לפונקציה?
8. כיצד מחזירים ערך מפונקציה?
9. מהו מקרו?
10. כיצד מחזירים ערך מטיפוס float?

תרגילים

1. כתוב תוכנית חידון שמציגה ארבע שאלות, כשכל שאלה ותשובה בפונקציה אחרת.
2. כתוב תוכנית שמזינה מספר ומשתמשת בפונקציה כדי לחשב ולהציג את החזקה הרביעית של המספר.

3. שנה את התוכנית שבתרגיל מספר 2 כך שחישוב החזקה הרביעית יתבצע בפונקציה, אך התוצאה תוחזר ל-main() לצורך הצגתה.

4. הסבר מה לקוי בתוכנית הבאה:

```
dothis()
{
    puts("This is first");
    main()
    return;
}
main()
{
    puts("This is second");
    return(0);
}
```

8

להשאיר את ההחלטה בידי המחשב

החל מפרק זה, כישורי התכנות שלך עומדים בפני זינוק אדיר קדימה. הנושאים בהם נעסוק בפרק זה ובפרקים הבאים אינם קשים יותר מאלו שכבר הכרת, אולם השילוב בינם לבין מה שכבר למדת, יציב אותך על-פני מדרגה גבוהה יותר מבחינת התחכום שבתכנות.

כמו כן, החל מפרק זה נפנה תשומת לב רבה יותר ללוגיקה של עיצוב התוכנית. ככל שכישוריך בשימוש בכתיב של שפת C ושפת ++C יגדלו, תבחין בשינוי הדגשים. במקום להתמקד בכל נקודה-פסיק ובכל תוחם, נתרכז יותר באלגוריתמים ובפתרון בעיות. תגלה שלא קיימת דרך "נכונה" אחת בלבד לכתוב תוכנית - ושבאפשרותך לבצע מטלה על ידי כתיבת תוכנית במספר דרכים שונות.

בפרק זה תלמד כיצד לגרום למחשב לקבל החלטות. במקום לבצע כל אחת מההוראות על-פי הסדר בו כתבת אותן, התוכנית תחליט, על-פי קריטריונים שתקבע, אילו הוראות לבצע. למעשה, תוכל להשתמש בטכניקה שתלמד בפרק זה לתיקון מרבית הבעיות הלוגיות שהצבענו עליהן בפרקים הקודמים.

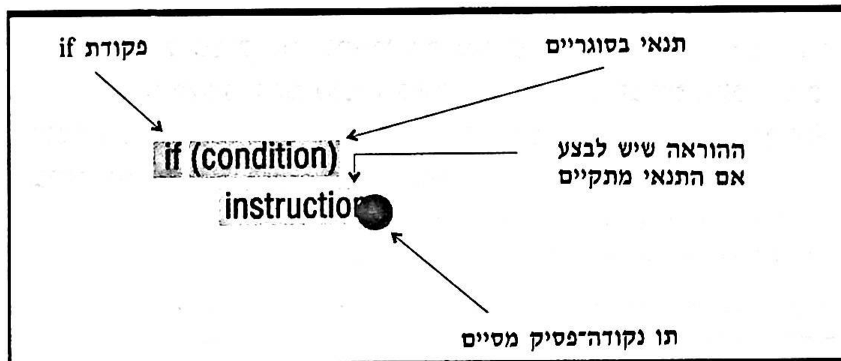
if - פקודה קטנה-גדולה

כל הקסם הזה מבוצע על ידי הפקודה הקטנה if - מלה קטנה, אך בעלת עוצמה גדולה. אנו משתמשים בפקודה if כדי להחליט אם יש לבצע הוראה מסוימת. מבנה הפקודה הוא כדלקמן:

```
if (condition)
instruction;
```

הפקודה אומרת למעשה את המשפט הבא: "אם התנאי מתקיים (אמת), בצע את ההוראה הבאה". (ראה תרשים 8.1). המחשב מבצע את ההוראה וממשיך, כרגיל, אל ההוראה הבאה אחרי פקודת ה-if.

תרשים 8.1
מבנה הפקודה if.



לעומת זאת, אם התנאי אינו מתקיים (שקר), המחשב מדלג על ההוראה המהווה חלק מפקודת ה-if, ועובר להוראה הבאה בתוכנית. קל לכתוב פקודת if, כל עוד מקפידים על הנקודות הבאות:

- הצב את התנאי בתוך סוגריים.

- אין להציב נקודה-פסיק אחרי התנאי - אלא רק אחרי ההוראה.
- אנו אומנם מתייחסים לפקודה כאומרת "אם התנאי מתקיים, אז בצע את ההוראה", אך המלה אז (then) אינה קיימת בשפת C.
- C היא שפת תכנות בעלת פורמט חופשי, ולפיכך - התנאי וההוראה יכולים להיות בשורה אחת. עם זאת, ההפרדה וכניסת הפסקה מקלים על קריאת התוכנית.

תנאים

התנאים בפקודות if מבצעים השוואה בין ערכים - משתנה או קבוע מושווים עם ערך מילולי או עם משתנה או קבוע אחרים. ההשוואה מתבצעת בעזרת אחד האופרטורים הבאים:

אופרטור	מובן	דוגמה
==	שווה	if (tax == 0.06)
>	גדול מ-	if (hours > 40)
<	קטן מ-	if (hours < 40)
>=	גדול או שווה	if (salary >= 10000)
<=	קטן או שווה	if (cost <= limit)
!=	שונה מ-	if (count != 1)

שים לב שכדי לבחון אם שני ערכים הם זהים, עליך להשתמש באופרטור ==, שני תווי שווה רצופים. אם תשתמש בתו שווה אחד בלבד, הקומפיילר יציג הודעת שגיאה. תו שווה אחד משמש רק להקצאת ערכים למשתנים.

פקודת if פשוטה עשויה להיראות כך:

```
if (time > 11)
    puts("Go home, it is after curfew.");
```

ההוראות אומרות למעשה את המשפט הבא: "אם המשתנה time גדול מ-11, אזי הצג את ההודעה הבאה". אם ערכו של המשתנה הוא 11 או קטן מ-11, ההודעה לא תוצג.

ניתן להשתמש בתנאי if כדי לבחון משתנים מספריים או משתנים מטיפוס char, אך לא מחרוזות. קטע התוכנית הבא, לדוגמה, יקומפל מבלי להציג הודעת שגיאה כלשהי, אך לא יציג את התוצאה הנכונה:

```
gets(name);
if (name == "Adam")
 puts("Call home");
```

מחרוזות הן משתנים מיוחדים, שנעסוק בהם ביתר פירוט בפרק 10.

עיין בתוכנית 8.1, תוכנית מלאה שעושה שימוש בפקודת if. זוהי גרסה מתוקנת של תוכנית שכבר פגשת בפרק קודם, המחשבת את סכומה של הזמנה, כולל מס הקנייה. בתוכנית זו, מוטל מס מותרות מיוחד על פריטים שמחירם עולה על $40,000. החישוב מבוצע בפקודה הבאה:

```
if (cost > 40000.00)
  luxury = cost * 0.005;
```

תוכנית 8.1: תוכנית לחישוב סכום הזמנות רכש, כולל מס מותרות

```
/*luxury1.c*/
main()
{

     float  cost, tax, luxury, total;
     luxury = 0.0;
     printf("Enter the cost of the item: ");
     scanf("%f", &cost);
     tax = cost * 0.06;
     if (cost > 40000.00)
          luxury = cost * 0.005;
     total = cost + tax + luxury;
     printf("The total cost is %.2f", total);
}
```

הפקודה אומרת, למעשה, את המשפט הבא: "אם ערכו של המשתנה cost גדול מ-$40,000, אזי הצב את הערך של cost כפול 0.5 אחוז במשתנה luxury". שתי ההוראות האחרונות בתוכנית מבוצעות בכל מקרה, בין אם מוטל מס מותרות ובין אם לאו, מכיוון שהן מופיעות אחרי תו הנקודה-פסיק המסיים את פקודת ה-if.

שים לב שבשלב מוקדם יותר בתוכנית מוקצה למשתנה luxury ערך התחלתי 0, אך לא מוקצה ערך התחלתי למשתנה tax. הסיבה לכך היא שהמשתנה tax יקבל תמיד ערך כתוצאה מהחישוב tax = cost * 0.06, בעוד שהמשתנה luxury מקבל ערך מחושב רק במקרה שמחיר הפריט גדול מ-$40,000. אם התנאי אינו מתקיים (שקר), החישוב כלל אינו מתבצע. אם לא היינו מקצים ל-luxury ערך התחלתי, היה מתווסף אל הסכום הכולל ערך בלתי-ידוע כלשהו. ככלל, כדאי להקצות ערכים התחלתיים למשתנים שערכיהם מחושבים בפקודת if, כמו גם למונים ולאוגרים.

כאשר אתה משתמש באופרטורים גדול מ-או קטן מ-, ודא שהם אכן מבצעים את התנאי הדרוש לך. בדוגמה הקודמת, מס המותרות מוטל רק אם מחיר הפריט הוא לפחות סנט

אחד יותר מ-$40,000, כלומר, החל מסכום של $40,000.01. אם מחיר הפריט $40,000 בדיוק, לא יוטל עליו מס המותרות. אם המס היה מוטל על פריטים שמחירם $40,000 ומעלה, התנאי המתאים היה כדלקמן:

```
if (cost >= 40000.00)
```

ההבדל בין < לבין =<, ובין > לבין => הוא אומנם עדין, אך הוא חיוני לצורך קבלת תוצאות מדויקות בתוכנית.

פקודות מרובות

פקודת if הבסיסית מבצעת הוראה אחת. אם ברצונך לבצע יותר מהוראה אחת כאשר התנאי מתקיים, עליך להשתמש ברמה נוספת של סוגריים מסולסלים. הסוגריים המסולסלים מסמנים את נקודת ההתחלה ונקודת הסיום של קבוצת הוראות, אותן יש לבצע כאשר התנאי מתקיים (אמת).

עיין לדוגמה בתוכנית 8.2, המדגימה ביצוע של שתי הוראות כאשר התנאי מתקיים. תו הסוגר המסולסל שאחרי התנאי מסמן את נקודת ההתחלה של קבוצת ההוראות, ואילו תו הסוגר המסולסל שאחרי ההוראה השנייה מסמן את נקודת הסיום של קבוצת ההוראות. הוראה כלשהי הכלולה בקבוצת ההוראות תתבצע רק כאשר התנאי מתקיים (אמת). השורות מסורגות כדי להבהיר לקורא שהן מהוות חלק מפקודת ה-if, אולם אין לכך כל משמעות מבחינת הקומפיילר.

תוכנית 8.2: הוראות מרובות בפקודת if

```
/*luxury2.c*/
main()
{
float  cost, tax, luxury, total;
luxury = 0.0;
printf("Enter the cost of the item: ");
scanf("%f", &cost);
tax = cost * 0.06;
if (cost > 40000.00)
{
    luxury = cost * 0.005;
    printf("The luxury tax is %.2f\n", luxury);
}
total = cost + tax + luxury;
printf("The total cost is %.2f", total);
}
```

הפקודה if...else

כאשר משתמשים בפקודת if כשלעצמה, מעוניינים רק בביצוע הוראות כאשר התנאי מתקיים (אמת). ניתן גם לבצע סדרת הוראות אחת כאשר התנאי מתקיים, וסדרה אחרת של הוראות כאשר התנאי אינו מתקיים (שקר). לדוגמה, נניח שברצוננו להדפיס את ההודעה "על פריט זה לא מוטל מס מותרות" עבור פריטים שמחירם נמוך מרמת המס. נוכל לעשות זאת בעזרת שתי פקודות if, כך:

```
if (cost > 40000.00)
{
      luxury = cost * 0.005;
      printf("The luxury tax is %.2f", luxury);
}
if (cost <= 40000.00)
      puts("There is no luxury tax for this item");
```

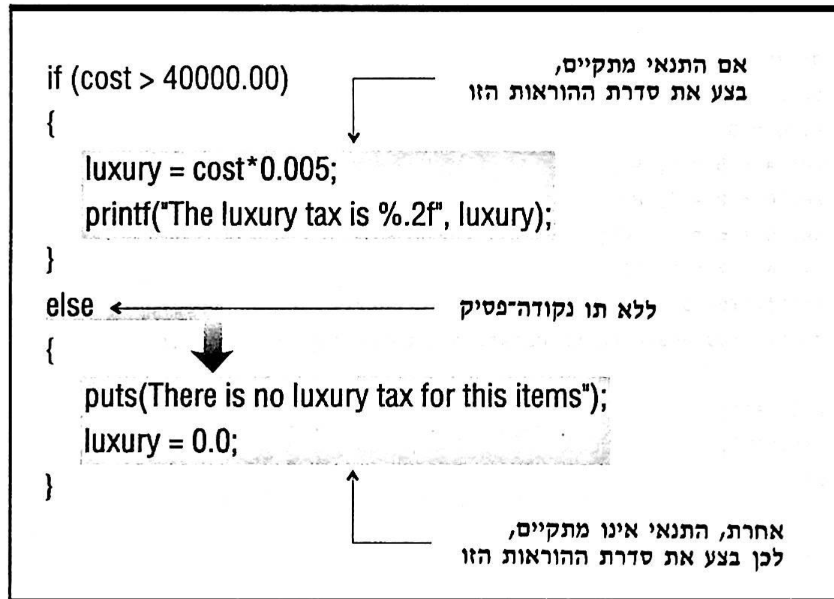
יחד עם זאת, קיימת חלופה יעילה יותר. ניתן לשלב את שתי ההוראות בפקודה אחת, מכיוון שהן למעשה שני מצבים אפשריים של משתנה אחד. ערכו של משתנה המחיר, cost, יכול להיות גבוה מ-$40,000, ויכול להיות נמוך או שווה ל-$40,000. אם תנאי אחד הוא שקר, הרי שהתנאי השני חייב להיות אמת. משלבים את ההוראות בעזרת הפקודה else:

```
if (condition)
  instruction;
else
  instruction;
```

המבנה הזה אומר למעשה את המשפט הבא: "אם התנאי הוא אמת, בצע את ההוראה הבאה; אחרת - בצע את ההוראה המופיעה אחרי הפקודה else". ההוראות המופיעות אחרי המלה else מבוצעות רק אם התנאי אינו מתקיים. אם ברצונך לבצע יותר מהוראה אחת, בכל אחד מהמקרים, הקף את ההוראות בסוגריים מסולסלים. יש להציב תו נקודה-פסיק אחרי כל הוראה, אך לא אחרי מלת המפתח else.

כדי להדפיס את ההודעה במקרה שלא מוטל על הפריט מס מותרות, יש לשנות את התוכנית, כפי שמודגם בתרשים 8.2. שים לב שכעת איננו חייבים להקצות למשתנה luxury ערך התחלתי בתחילת התוכנית, מכיוון שהפקודה if לוקחת עתה בחשבון את כל המצבים האפשריים.

תרשים 8.2 גרסה שונה של התוכנית תוך שילוב מלת המפתח else.



ביקור חוזר בתוכנית החידון

בפרק 7 ראינו מספר תוכניות שמציגות שאלות ותשובות. מכיוון שאז עדיין לא הכרת את הפקודה if, לא יכולת להעניק נקודות או ציונים. הענקת נקודות היא פשוט עניין של השוואת התשובה הנכונה עם התשובה שהמשתמש נותן באמצעות המקלדת, כפי שמודגם בתוכנית 8.3.

תוכנית זו עושה שימוש בפונקציה כדי להציג את השאלה, להזין את התשובה, לקבוע אם התשובה נכונה ולמנות את מספר התשובות הנכונות והתשובות השגויות. הן השאלה והן התשובה הנכונה מועברות אל הפונקציה - השאלה היא מחרוזת מילולית, ואילו התשובה היא מספר שלם. התוכנית כתובה באופן שניתן יהיה להוסיף שאלות על ידי החדרת קריאות לפונקציה, כדוגמת:

```
ask("9 + 5 = ", 14);
```

תוכנית 8.3: תוכנית חידון שמעניקה נקודות

```
/*score*/
int correct, wrong;
main()
{
```

```
    char question[15];
    int answer;
    correct = 0;
    wrong = 0;
    ask("4 + 5 = ", 9);
    ask("6 + 2 = ", 8);
    ask("5 + 5 = ", 10);
    ask("4 + 7 = ", 11);
    printf("You answered %d question correctly.\n", correct);
    printf("You answered %d question incorrectly.\n", wrong);
}
ask(quest, ans)
char quest[15];
int ans;
{
    int guess;
    printf("%s", quest);
    scanf("%d", &guess);
    if (guess == ans)
        ++correct;
    else
        ++wrong;
    return(0)
}
```

אופרטורים לוגיים

תנאי if שראינו עד עתה בוחנים רק משתנה אחד וערך אחד. כלומר, רק תנאי אחד צריך להתקיים כדי שהביטוי יוערך כביטוי אמת. במציאות, תוכניות מחשב חייבות לעתים קרובות לבחון יותר מערך אחד.

עיין בתוכנית 8.4, שמניחה מראש שלא על כל פריט שאתה מוכר מוטל מס. במקום להוסיף את מס הקנייה באופן אוטומטי לכל פריט, התוכנית שואלת תחילה אם הפריט המסוים חייב במס, ואז - אם אכן הפריט חייב במס, היא מוסיפה למחירו מס קנייה בשיעור 6 אחוזים.

תוכנית 8.4: לוגיקה המבוססת על קלט מהמשתמש

```
/*iftax.c*/
main()
{
     int taxable;
     float  cost, tax;
     tax = 0.0;
     printf("Enter the cost of the item: ");
     scanf("%f", &cost);
     printf("Enter Y if the item is taxable, N if nontaxable: ");
     taxable=getchar();
     if (taxable == 'Y')
          tax = cost * 0.06;
     printf("\nThe total due is %f", cost + tax);
}
```

אם התוכנית נראית לך נכונה, עיין בה שוב - היא סובלת מפגם משמעותי. התוכנית מניחה שהמשתמש יקליד את האות הגדולה Y אם הפריט חייב במס. במקרה שהמשתמש מקליד אות y קטנה, התוכנית מסיקה שהפריט אינו חייב במס, מכיוון שפקודת ה-if בוחנת רק עבור Y גדולה.

במצבים כאלה - יש לבחון עבור שני סוגי הקלט האפשריים - Y או y. ניתן אומנם לעשות זאת בעזרת שתי פקודות if, אך במקום זאת נשתמש באופרטור הלוגי או - || - כך:

```
if (taxable == 'Y' || taxable == 'y')
```

הוראה זו אומרת למעשה את המשפט הבא: "אם taxable שווה Y או taxable שווה y". לפיכך, הפריט ייחשב כחייב במס אם אחד משני התנאים הוא אמת. במקרה ששני התנאים אינם אמת - כלומר, המשתמש הקליד תו אחר כלשהו שאיננו Y או y - הפריט לא יחויב במס. התנאי כולו מוקף בזוג סוגריים אחד, והמשתנה אותו בוחנים - taxable - מופיע פעמיים. אם נכתוב את התנאי כך: (taxable == 'Y' || 'y') - תיווצר שגיאת קומפילציה.

הערות בנושא הפונקציה ()getchar

בקומפיילרים מסוימים (אך לא בקומפיילר PCC שעל-גבי התקליטון המצורף לספר), הפונקציה ()getchar פועלת דרך חוצץ, או זיכרון זמני. פירוש הדבר שהתו המוזן מאוחסן בזיכרון המחשב עד שהמשתמש מקיש על מקש Enter. אם כך פועל הקומפיילר שברשותך, עיין בתיעוד הנלווה לקומפיילר לגבי הפונקציות ()getch או ()getche. פונקציות אלה אינן פועלות בדרך כלל דרך חוצץ, וניתן להזין תו בודד בעזרתן מבלי להזדקק להקשה על Enter.

בנוסף, שימוש בפונקציה ()getchar אחרי פקודת הקלט ()scanf עלול לגרום לבעיה. כפי שהוסבר בפרק 5, אם לא מזינים את הנתונים המתאימים לפונקציה ()scanf, תווים אחדים עלולים להישאר בחוצץ הקלט. הפונקציה ()getchar עשויה לקרוא את אחד התווים האלה במקום את התו שהקלדת. ניתן להימנע מכך על ידי שימוש בפקודה ()gets או בפקודת קלט לתו יחיד אחרת, שאינה פועלת דרך חוצץ, כמו ()getch או ()getche, במידה שהן זמינות בקומפיילר שברשותך.

אפשרות נוספת היא לרוקן את חוצץ הקלט לפני השימוש בפונקציה ()getchar. כדי לעשות זאת, הוסף את הפקודה <include <stdio.h# בתחילת התוכנית. לאחר מכן, החדר את הפקודה (fflush(stdin לפני הפונקציה ()getchar. הפקודה ()fflush מרוקנת (מסלקת) את התווים שנותרו בחוצץ הקלט התקני.

קיימים שלושה אופרטורים לוגיים: || מסמן את האופרטור OR (או), && מסמן את האופרטור AND (וגם), והתו ! מסמן את האופרטור NOT (לא). כאשר אתה משתמש באופרטור OR, אחד משני התנאים (או שניהם) חייב להיות אמת כדי שביטוי ה-if יהיה אמת. כאשר משתמשים באופרטור AND, שני התנאים חייבים להיות אמת. כאשר משתמשים באופרטור NOT, על התנאי להיות שקר כדי שביטוי ה-if יבוצע.

ניתן להשתמש באופרטור && ובאופרטור || כדי לבחון משתנה אחד לעומת שני ערכים, כפי שעשינו זה עתה, או כדי לבחון שני משתנים שונים. לדוגמה, נניח שעליך לכתוב תוכנית שמזינה את משכורתו של המשתמש ואת מספר בני המשפחה התלויים בו, כמו תוכנית 8.5.

תוכנית 8.5: בחינת שני משתנים

```
/*twovars.c*/
main()
{
     int depents;
     float income;
     puts("Please enter your income");
     scanf("%f", &income);
     puts("Please enter the number of dependents");
     scanf("%d", &depents);
     if (income < 20000 && depents > 2)
        puts("You owe no income tax");
}
```

תנאי ה-if בוחן שני משתנים, income ו-depents. ההכנסה (income) חייבת להיות קטנה מ-$20,000, **וגם** מספר התלויים (depents) חייב להיות גדול משניים - כדי שהתנאי יתקיים וההודעה תוצג. אם אחד משני התנאים אינו מתקיים, כמו במקרה של בן משפחה אחד או הכנסה של $20,001, לא תבוצע הוראת ה-()puts.

יש לנקוט זהירות בשימוש באופרטור &&, ולוודא שהוא אכן משיג את מטרת התנאי. לדוגמה, אל תשתמש לעולם באופרטור && כדי לבחון אם למשתנה כלשהו יש שני ערכים. התנאי

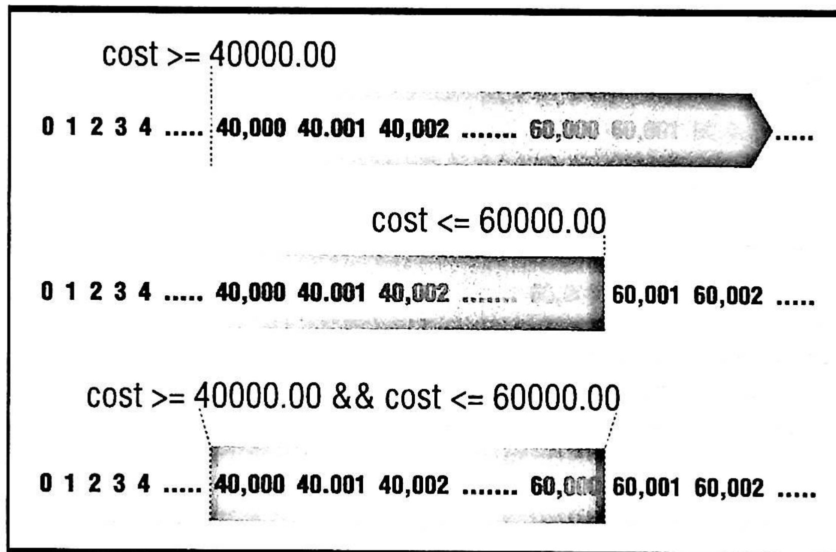
```
if (taxable == 'Y' && taxable == 'y')
```

לעולם לא יתקיים. משתנה אינו יכול להיות שווה לשני ערכים שונים. לעומת זאת, ניתן לבחון משתנה אחד כדי לקבוע אם הוא בתוך טווח של ערכים, או מחוץ לטווח.

כדוגמה, נניח שמס המותרות מוטל רק על פריטים שמחירם נע בין $40,000 לבין $60,000. כדי לבחון זאת נציב את התנאי הבא:

```
if (cost >= 40000.00 && cost <= 60000.00)
```

כדי שביטוי ה-if יהיה אמת, שני התנאים חייבים להתקיים, בגלל האופרטור and. כדי ששני התנאים יתקיימו, מחיר הפריט צריך להיות גדול או שווה ל-$40,000 וגם קטן או שווה ל-$60,000, בעת ובעונה אחת - כלומר, בתוך טווח המחירים, כפי שמודגם בתרשים 8.3.



תרשים 8.3
שימוש באופרטור לוגי כדי לקבוע אם ערך כלשהו נמצא בתוך טווח ערכים.

אם היית משתמש כאן, בטעות, באופרטור **או**, כל ערך של המשתנה cost היה מקיים את התנאי. כדי לקבוע אם ערך כלשהו נמצא מחוץ לטווח ערכים נקוב, עליך להשתמש באופרטור **או** ולשנות את מיקומם של האופרטורים גדול-שווה וקטן-שווה. הביטוי הבא בוחן אם הערך של המשתנה income נמצא מחוץ לטווח, על ידי הצבת השאלה האם הערך קטן-שווה לגבול התחתון של הטווח, או גדול-שווה לגבול העליון:

```
if (income <= 20000.00 || income >= 60000.00)
    puts("You are not middle class");
```

האופרטור **לא** (NOT) נקרא אופרטור *יחידני* מכיוון שהוא פועל על פריט אחד בלבד - שם של משתנה או של קבוע. אם נקבע שהערך 0 נחשב לשקר (false), כל ערך שאינו אפס - בין אם הוא מספר שלילי או מספר חיובי - ייחשב לאמת. ההוראות הבאות, לדוגמה, יציגו את ההודעה "Count is false", מכיוון שערכו של המשתנה count הוא 0:

```
int count;
count = 0;
if (!count) puts ("count is false");
```

התנאי פועל בצורה זהה לביטוי if (count == 0). באופן דומה, ההוראות הבאות יציגו את ההודעה "Count is true" מכיוון שערכו של count שונה מאפס:

```
int count;
count = 1;
if (count) puts ("Count is true");
```

התנאי זהה לביטוי (count != 0) if, כתיב מקוצר לבדיקה אם הערך שונה מאפס. בפרקים הבאים תלמד כיצד להשתמש באופרטורים יחידניים.

פקודות if מקננות

אחרי תנאי if או פקודת else יכולה לבוא כל הוראה שהיא. ההוראה יכולה לשמש לקלט או לפלט של ערך, לבצע פעולה מתמטית או לקרוא לפונקציה. ההוראה יכולה גם להיות פקודת if נוספת. פקודת if שנתונה בתוך פקודת if אחרת נקראת *מקננת* (nested). לדוגמה, בקבוצת ההוראות הבאה, פקודת ה-if השנייה מקננת בתוך הראשונה:

```
if (income > 100000)
 if (status == 'S')
  taxrate = 0.35;
```

תנאי ה-if השני נבחן רק כאשר תנאי ה-if הראשון מתקיים, ומכאן שהמשתנה taxrate מקבל ערך רק במקרה ששני התנאים הם אמת. ניתן לכתוב זאת גם בעזרת אופרטור לוגי, כך:

```
if (income > 100000 && status == 'S')
  taxrate = 0.35;
```

שתי הדוגמאות מבצעות מטלה זהה, אולם צורת הכתיבה השנייה, תוך שימוש באופרטור הלוגי &&, ברורה יותר. אין צורך לפענח מה עושה כל ביטוי if בנפרד; קריאת הביטוי מסבירה את משמעותו: "אם income גדול מ-100,000 וגם status שווה S, אזי taxrate שווה 0.35".

ככלל, ניתן להחליף כל שתי פקודות if מקננות עוקבות - בביטוי אחד הכולל את האופרטור הלוגי &&. כלל נוסף: כדאי להימנע מפקודות if מקננות, מכיוון שהן עלולות להביא למצבים בלתי-ברורים ולקוד קשה לקריאה. לדוגמה, עיין בתוכנית הבאה:

```
main()
{
float income;
scanf("%f", &income);
if (income >= 20000.00)
  if (income <= 100000.00)
   puts("Your tax rate is 22%");
else
     puts (You earn less than $20,000 so your rate is 15%");
}
```

מקריאת התוכנית נראה שהיא אומרת כדלקמן: "אם income גדול מ-$20,000 וגם קטן מ-$100,000, הדפס את ההודעה הראשונה, אולם אם income אינו גדול מ-$20,000, אזי הדפס את ההודעה השנייה".

לרוע המזל, התוכנית תתקמפל ללא שגיאות, אך לא תציג את התוצאות הנכונות. אם תקליד ערך קטן מ-$20,000 לא תתבצע אף אחת משתי הוראות ה-puts(), ואם תקליד ערך גדול מ-$100,000, ידווח שיעור המס כ-15% בלבד.

פקודת ה-else מקושרת לביטוי ה-if הקרוב אליה ביותר, ללא קשר להזחת השורות בתוכנית. בתוכנית הזו, הזחת השורות מטעה את הקורא לחשוב שפקודת ה-else מקושרת לתנאי ה-if הראשון, אך למעשה, ה-else מקושרת אל פקודת ה-if השנייה - ומתבצעת רק במקרה ש-income גדול מ-20,000 אך אינו גדול מ-100,000.

כדי להשיג את הכוונה המקורית, יש לכתוב את ההוראות כך:

```
if (income >= 20000.00)
  {
  if (income <= 100000.00)
   puts("Your tax rate is 22%");
  }
else
    puts("You earn less than $20,000 so your rate is 15%");
```

הסוגריים המסולסלים מבודדים את פקודת ה-if המקננת ומאלצים אותה להסתיים בתוחם הסוגר. הפקודה else מקושרת עתה לפקודת ה-if הראשונה.

דרך טובה יותר לכתוב את קטע התוכנית הזה היא בעזרת פקודת if אחת, כך:

```
if (income >= 20000.00 && income <= 100000.00)
   puts("Your tax rate is 22%");
else
    puts("You earn less than $20,000 so your rate is 15%");
```

בצורה זו נמנעת אי-הבהירות הנובעת מפקודות if מקננות וכפל סוגריים מסולסלים.

תיקנו אומנם את הלוגיקה של התוכנית, אולם היא עדיין סובלת מפגם משמעותי. התוכנית מציגה הודעה רק במקרה שההכנסה (income) שווה או קטנה מ-$100,000. תוכנית שבנויה כהלכה חייבת לקחת בחשבון את כל המצבים האפשריים. לדוגמה, אם תקליד ערך הכנסה של $150,000, לא תוצג כל הודעה. האם פירוש הדבר שאינך חייב כלל במס!

פתרון אפשרי אחד, המוצג בתוכנית 8.6, עושה שימוש בפקודות if...else מקננות כדי להתייחס לכל רמות ההכנסה האפשריות. ודא שברור לך האופן בו פקודות if אלה מקננות

אחת בתוך השנייה. אם התנאי הראשון הוא אמת (income < 20000.00), מתבצעת הוראת ה-()puts הראשונה, והתוכנית מדלגת על יתר ההוראות. מילת המפתח else הראשונה מקושרת לפקודת ה-if הראשונה, ולכן התנאי השני (income < 100000.00) מתבצע רק אם התנאי הראשון אינו מתקיים.

תוכנית 8.6: שימוש בפקודות if מקננות כדי להתייחס לכל התנאים

```
/*brackets.c*/
main()
{
     float income;
     printf("Enter your income: ");
     scanf("%f", &income);
     if (income < 20000.00)
          puts("Your tax rate is 15%");
     else
          if (income < 100000.00)
               puts("Your tax rate is 22%");
          else
               puts("Your tax rate is 35%");
}
```

שים לב שקיימים שלושה מצבים אפשריים, אולם רק שני תנאי if. כשמשתמשים בשילובי if...else עוקבים, זקוקים לתנאי אחד פחות ממספר המצבים האפשריים. אם המצב הראשון אינו מתקיים, והמצב השני אינו מתקיים, אזי המצב השלישי חייב להתקיים - אין צורך בפקודת if כדי לבחון אותו. אילו היו ארבעה מצבים אפשריים, היינו זקוקים לשלושה שילובי if...else ולפקודת else רביעית מסיימת.

המבנה של switch/case/default

כאשר קיימים ארבעה מצבים אפשריים, או יותר, הלוגיקה של פקודות if...else מקננות עלולה להיעשות מסובכת ומטעה. הפקודה switch משמשת כחלופה במקרים כאלה. switch מציגה מבנה דמוי-תפריט, שבו רשומים כל המצבים האפשריים וההוראות שיש לבצע בכל אחד מהמצבים, כפי שמדגימה תוכנית 8.7.

תוכנית 8.7: שימוש בפקודה switch

```
/*switch.c*/
main()
{
     int answer;
     puts("The C language is: \n");
     puts("1. A language spoken in the south of France\n");
     puts("2. Used only for writing large computer programs\n");
     puts("3. A compiled language easily ported to various
systems\n");
     puts("4. None of the above\n");
     puts("Enter your answer from 1 to 4\n");
     answer= getchar();
     putchar('\n');
     switch (answer)
     {
     case '1':
      puts("Sorry, you are incorrect.\n");
      puts("In the south of France, they speak Pascal\n");
      break;
     case '2':
      puts("Sorry. The C language can be used to write programs
\n");
      puts("of all types and sizes.\n");
      break;
     case '3':
      puts("Very good, you are correct\n");


      puts("The C language is compiled, and it can be used on
a\n");
      puts("variety of computer systems.\n");
      break;
     case '4':
      puts("Sorry. Only number 3 is correct.\n");
      break;
     default:
      puts("You responded with a letter or a number other than 1
to 4\n");
     }
}
```

הפקודה switch זקוקה לארגומנט המכיל משתנה מטיפוס מספר שלם או מטיפוס char. אחרי הפקודה מופיעות הוראות הנתונות בתוך סוגריים מסולסלים, המורכבות מסדרה של פקודות case. כל פקודת case מבצעת הוראות המבוססות על ערך אפשרי כלשהו של המשתנה. הערך חייב להיות מבוטא כמספר שלם, כערך מילולי הנתון בגרשיים או כשמו של קבוע מטיפוס מספר שלם או מטיפוס char.

הביטוי :'1' case, לדוגמה, מורה לקומפיילר לבצע את ההוראות הבאות אחרי הביטוי, בתנאי שערכו של משתנה ה-switch הוא "1". במקרה שהערך אינו "1", הקומפיילר מדלג אל ביטוי ה-case הבא ובוחן אותו.

כאשר הערך מתאים לתנאי ה-case, מבוצעות ההוראות המופיעות אחרי ה-case. הביטוי break המופיע בסופה של כל סדרת case מעביר את השליטה אל סופה של הפקודה switch. מכאן נובע, שאם מתבצעת סדרת case אחת, הקומפיילר מתעלם משאר סדרות ה-case ופקודת switch מסתיימת.

אם לא מתקיים אף אחד מתנאי ה-case, מבוצעות ההוראות שבקטע ה-default. בשיטה זו קל לתת מענה לכל אפשרויות הקלט. אין צורך בפקודת break עבור תנאי ה-default, מכיוון שזהו תמיד הקטע האחרון ברוטינת ה-switch. כלול בתוכנית קטע default גם אם אתה סבור שביכולתך לתת מענה לכל התנאים האפשריים בעזרת סדרות ה-case השונות.

אם השמטת ביטוי break, המחשב יבצע את ההוראות המופיעות אחרי case שהוא אמת, עד לביטוי ה-break הבא. ניתן להשתמש בתכונה זו כדי לבצע סדרת הוראות אחת עבור יותר מ-case אחד, כמו בדוגמה להלן:

```
case 'Y':
case 'y': puts("You responded with a yes");
     break;
case 'N':
case 'n': puts("You responded with a no");
     break;
```

בדיקת משתנים מטיפוס float ומחרוזות

מכיוון שהערך בביטוי case חייב להיות מטיפוס מספר שלם או מטיפוס char, אינך יכול לכלול בתוכנית ביטויי case כגון :case 12.87, או :"case "Adam. במחרוזות נדון בפרק 10, אך אם עליך לבחון ערכים מטיפוס float, יש להמיר אותם תחילה לפורמט של מספר שלם או של ערך מילולי מטיפוס char.

במרבית המקרים ניתן להשתמש בסדרה של פקודות if...else, כמו בתוכנית 8.8. פקודות if...else מקצות ערך מספר שלם למשתנה ששמו level, בהתבסס על ערך מטיפוס float המוזן לתוך המשתנה income.

תוכנית 8.8: שימוש בפקודות *if* מקננות כדי לטפל בערכים מטיפוס float

```
/*switchf.c*/
main()
{
     float income;
     char level;
     printf("Enter your income: ");
     scanf("%f", &income);
     if (income <= 20000.00)
       level = '1';
     else
       if (income <= 60000.00)
          level = '2';
       else
           if (income <= 120000.00)
             level = '3';
             else
             level = '4';
     switch (level)
     {
     case '1':
      puts("Taxrate = 15%");
      break;
     case '2':
      puts("Taxrate = 28%");
     break;
     case '3':
     puts ("Taxrate = 32%");
     break;
     case '4':
     puts ("Taxrate = 36%");
     break;
     default:
     puts("You entered an invalid entry");
     }
}
```

ממבט ראשון זה נראה טיפשי למדי להשתמש בסדרה של פקודות if...else וברוטינת switch, כאשר אפשר מלכתחילה לכתוב את כל ההוראות בפקודות if...else. בתוכניות פשוטות, כמו זו שבדוגמה, ניתן אכן לעשות זאת. אולם כאשר ההוראות שיש לבצע עבור כל case מורכבות יותר, וכוללות בעצמן פקודות if מקננות, המבנה של רוטינת switch עדיף. ביטויי ה-case מראים בבירור כמה מצבים נבחנים, ואילו הוראות מבוצעות עבור כל מצב. "בדק הבית" הכרוך בהקצאת ערכים מילוליים בעבודה עם סדרות נפרדות של פקודות if...else מצדיק את הטרחה הנוספת.

עיצוב התוכנית

כפי שניתן לראות, ישנן מספר דרכים לכתוב אותה תוכנית - תוך שימוש בכמה פקודות if, בפקודות if מקננות, באופרטורים לוגיים או בפקודת switch. תוכנית 8.9, למשל, מדגימה כיצד ניתן לבצע את תוכנית החידון (שבגרסה הקודמת עשתה שימוש בפקודת switch), בעזרת הוראות if...else מקננות. הבחירה תלויה בסגנונך האישי. איש לא יוכל לטעון שהדרך שבחרת בה אינה נכונה, כל עוד התוכנית מבצעת את כל מה שהיא אמורה לבצע. מובן שאם התוכנית מביאה לתוצאות שגויות - היא ללא ספק פגומה - אך אין זה בהכרח בגלל שבחרת מבנה שאינו נכון.

תוכנית 8.9: תוכנית החידון - שימוש בפקודות if...else מקננות

```
     /*quiz4.c*/
     main()

{
     int answer;
     puts("The C language is: \n");
     puts("1. A language spoken in the south of France\n");
     puts("2. Used only for writing large computer programs\n");
     puts("3. A compiled language easily ported to various
systems\n");
     puts("4. None of the above\n");
     puts("Enter your answer from 1 to 4\n");
     answer= getchar();
     putchar('\n');
     if (answer=='1')
      {
      puts("Sorry, you are incorrect.\n");
      puts("In the south of France, they speak Pascal\n");
     }
```

```
      else
       if (answer=='2')
         {
         puts("Sorry. The C language can be used to write
programs \n");
         puts("of all types and sizes.\n");
         }
        else
          if (answer=='3')
           {
            puts("Very good, you are correct\n");
            puts("The C language is compiled, and it can be used
on a"\n);
           puts("variety of computer systems.\n");
           }
          else
             if (answer=='4')
                 puts("Sorry. Only number 3 is correct.\n");
             else
     puts("You responded with a letter or a number other than 1
to 4\n");
}
```

בחר בשיטה הנוחה לך, שתיצור תוכנית ברורה ומובנת ככל האפשר. לעתים השיטה הטובה ביותר מצריכה שימוש במספר הרב ביותר של הוראות - התוכנית הקצרה ביותר אינה בהכרח גם הטובה ביותר. בכל שיטה שתבחר עבור תוכנית מסוימת, הקפד תמיד לוודא את תקפות הקלט.

בדיקת תקפות הקלט

אל תניח מראש שהמשתמש יקליד את הערך הנכון, או אות קטנה או גדולה, בהתאם לצורך. לדוגמה, בפרק 7 ראינו תוכנית שמניחה מראש שהמשתמש יקליד את שיעור ההנחה למוצר כמספר עשרוני. תוצאת החישוב תהיה שגויה אם המשתמש יקליד את הערך 5 במקום 0.05 לציון חמישה אחוזים. אחת הדרכים לפתור את הבעיה היא להוסיף את הפקודה הבאה:

```
if (mrkdown > 1)
     mrkdown = mrkdown / 100;
```

פקודת ה-if תמיר את הקלט מ-5 ל-0.05.

בפרק 6 ראינו תוכנית שמחשבת את משכורתו של עובד. כזכור לך, במקרה שהעובד צבר פחות מ-40 שעות עבודה שבועיות, התוכנית הפחיתה, למעשה, את התשלום המגיע לפי תעריף כפול לכל שעת עבודה חסרה. תוכנית 8.10 מתקנת את הפגם. פקודת ה-if כוללת שתי סדרות של הוראות - סדרה אחת מתבצעת כאשר העובד צבר 40 שעות עבודה לכל היותר, והסדרה השנייה מתבצעת כאשר מספר השעות גדול מ-40. לכל משתנה שיוצג מוקצה ערך באחת משתי הסדרות, כדי שלא יוצגו ערכים בלתי-ידועים.

תוכנית 8.10: תוכנית שכר מתוקנת

```
/*allhours.c*/
main()
{
     float rate, hours, total, regular, extra, d_time, overtime;
     printf("Enter your hourly rate of pay: ");
     scanf("%f", &rate);
     printf("Enter the number of hours you worked: ");
     scanf("%f", &hours);
     d_time=rate * 2;
     if (hours <= 40)
     {
          regular = hours * rate;
          extra = 0.0;
          overtime = 0.0;
          total = regular;
     }
     else
     {
          regular = 40 * rate;
          extra = hours - 40;
          overtime = extra * d_time;
          total = regular + overtime;
     }
     printf("Your regular weekly salary is %.2f\n", regular);
     printf("You  worked %.2f overtime hours\n", extra);
     printf("Your overtime rate is $%.2f\n", d_time);
     printf("Your overtime pay is %.2f\n", overtime);
     printf("Your total pay is %.2f\n", total);
}
```

התעריף הכפול, המיוצג על ידי המשתנה dtime, מחושב לפני הוראת ה-if, מכיוון שהתעריף הכפול תקף לגבי כל העובדים, גם עבור מי שלא עבד בפועל למעלה מ-40 שעות בתקופה הנוכחית. מכיוון שחישוב זה אינו מבוסס על קיום או אי-קיום של תנאי כלשהו, הוא מחושב מחוץ לפקודת ה-if.

עם זאת, הפתרונות שהצגנו לשתי הבעיות אינם מלאים. המשתמש עדיין יכול להקליד מספר שלילי עבור שיעור ההנחה, או ערך בלתי סביר, כמו 2500, עבור מספר שעות העבודה השבועיות. בפרק הבא תלמד כיצד לפתור באופן סופי את בעיית הקלט.

שאלות

1. מהו ההבדל בין הסימן = לבין הסימן ==?
2. כיצד מבצעים יותר מביטוי אחד כאשר תנאי כלשהו מתקיים?
3. מהי מטרתה של פקודת else?
4. כיצד בודקים אם מספר כלשהו נמצא בתוך טווח ערכים נתון?
5. הסבר את אופן השימוש בפקודות if "מקננות".
6. במה שונה פקודת switch מפקודת if?
7. כיצד ניתן להשתמש בערך מטיפוס float בפקודת switch?
8. הסבר כיצד ניתן להשתמש בפקודת if כדי לבדוק את תקפות הקלט.

תרגילים

1. כתוב תוכנית שמזינה מספר כלשהו ומדווחת אם המספר הוא זוגי או אי-זוגי.
2. כתוב תוכנית שמזינה מספר כלשהו ומדווחת אם המספר נמצא בטווח הערכים שבין 1 ל-100.
3. כתוב תוכנית שמזינה מספר שלם ומדווחת לאיזה טווח מספרים הוא משתייך: קטן מ-0, מ-0 עד 50, 51 עד 100, 101 עד 150 או גדול מ-150.
4. כתוב תוכנית המנחה את המשתמש להקליד מספרים לתוך המשתנים lownum ו-highnum. ערכו של lownum חייב להיות קטן מערכו של highnum. אם לא כך הם פני הדברים, על התוכנית להחליף בין המספרים ולהציב את הערך הקטן במשתנה lownum ואת הערך הגדול במשתנה highnum. הצג את הערכים כדי לוודא שהפעולה בוצעה כהלכה.



5. הסבר מה לקוי בתוכנית הבאה:

```
main()
{
int age;
printf(Enter your age);
scanf("%f", &age);
if age < 18 then
    puts("You cannot vote");
else
    if age > 18 then
        puts("You can vote");
}
```

9

פעולה חוזרת

תוכנית מתחילה ומסתיימת, אך אין פירוש הדבר שעליה לבצע פעולה כלשהי פעם אחת בלבד. ייתכן שעליך לטפל ביותר מהזמנה אחת, או שעליך לבצע כמה חישובים שונים. ייתכן שתרצה להמשיך ולבקש קלט מהמשתמש, עד שיקליד ערך הנמצא בתוך טווח הערכים הנכון.

שפת C ושפת ++C כוללות שלושה מבנים מובדלים, הנקראים לולאות, לצורך טיפול בפעולות חוזרות:

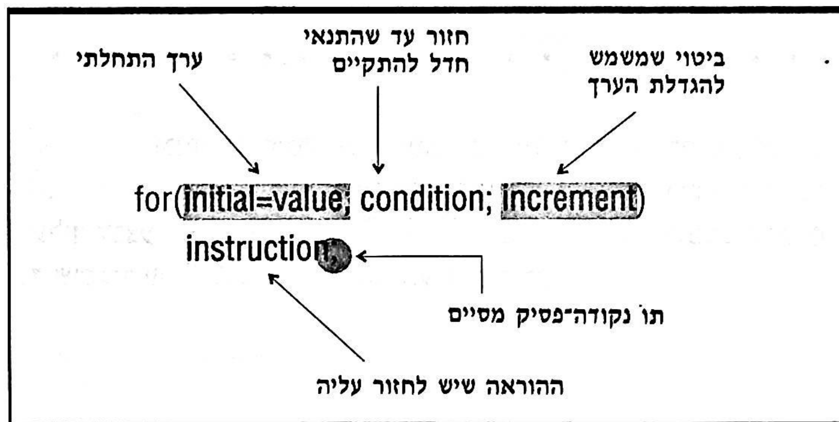
- לולאת for
- לולאת do...while
- לולאת while

כל אחד ממבנים אלה יכול לחזור על הוראה, על סדרה של הוראות או אף על תוכנית שלמה.

שימוש בלולאת for

השתמש בלולאת for כאשר ידוע לך המספר המדויק של חזרות שברצונך לבצע. מבנה הלולאה מודגם בתרשים 9.1. שים לב שאת תו הנקודה-פסיק המסיים יש להציב אחרי ההוראה, ולא אחרי הפרמטר של הביטוי for. שלושת מרכיבי הפרמטר שבתוך הסוגריים חייבים להיות מופרדים בנקודה-פסיק.

תרשים 9.1
המבנה של לולאת for.



לדוגמה, התוכנית הבאה משתמשת בלולאת for כדי להציג על המסך את המספרים 1 עד 10, שורה אחר שורה:

```
main()
{
    int repeat
    for(repeat=1; repeat <=10;repeat++)
        printf("%d\n", repeat);
}
```

הלולאה נשלטת על ידי משתנה ששמו repeat, הנקרא *משתנה האינדקס*. באפשרותך להעניק למשתנה האינדקס כל שם שתרצה, אך המשתנה חייב להיות מטיפוס מספר שלם. הפרמטר של for מתחלק למקטעים הבאים:

repeat = 1	מאתחל את הערך ההתחלתי של המשתנה repeat.
repeat<=10	קביעת תנאי החזרה: כל עוד repeat קטן-שווה ל-10.
repeat++	מגדיל את ערכו של repeat אחרי כל חזרה.

עם כל חזרה על הלולאה התוכנית מציגה את ערכו הנוכחי של המשתנה repeat.

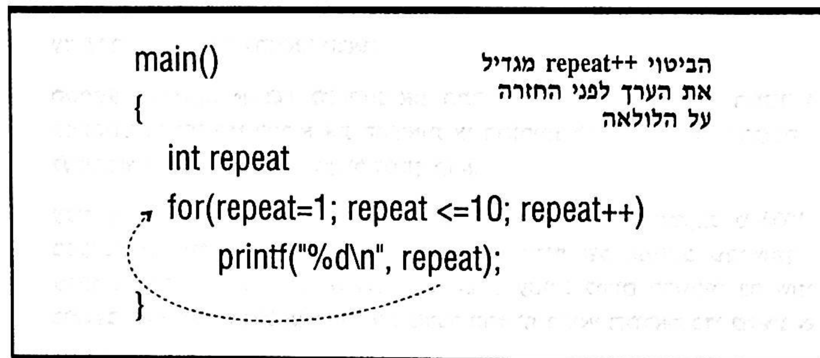
הערות ++C

בשפת ++C ניתן להכריז על משתנה האינדקס בתוך הפרמטר, כך:

```
for(int repeat=1; repeat <=10;repeat++)
```

כאשר התוכנית נתקלת בלולאה בפעם הראשונה, היא קובעת את ערכו של המשתנה repeat ל-1. לאחר מכן היא בודקת את התנאי כדי לראות אם הערך של repeat קטן או שווה ל-10. מכיוון שהתנאי מתקיים, התוכנית מבצעת את ההוראה הקשורה ללולאה, כלומר - מציגה את הערך של המשתנה.

אחרי ביצוע ההוראה, התוכנית מגדילה את ערכו של המשתנה ב-1, ובודקת שוב את התנאי (כפי שמודגם בתרשים 9.2). מכיוון שהתנאי עדיין אמת, התוכנית מבצעת את ההוראה פעם נוספת ומציגה את ערכו הנוכחי של המשתנה. התהליך חוזר על עצמו עד שהערך של המשתנה גדל ל-11. במצב זה התנאי repeat <= 10 כבר אינו מתקיים, ולכן ההוראה אינה מבוצעת והלולאה נפסקת.



תרשים 9.2
התנאי נבדק בסיום של כל לולאה.

בדוגמה הקודמת ההוראה עשתה שימוש בערכו של המשתנה. אין הכרח לכתוב כך את ההוראה. לדוגמה:

```
main()
{
      int repeat
      char letter
      puts("Please enter 10 characters");
      for(repeat=1; repeat <=10;repeat++)
            letter=getchar();
}
```

תוכנית זו מבצעת את ההוראה ()getchar עשר פעמים, פעם אחת עבור כל חזרה על הלולאה, כשהערך של repeat גדל מ-1 ל-11. משתנה האינדקס משמש רק לצורך קביעת מספר החזרות. ניתן גם לכתוב את הוראת ה-for בצורה הבאה, עם אותן התוצאות:

```
      for(repeat=101; repeat <=110;repeat++)
      letter=getchar();
}
```

גם בצורת כתיבה זו יוזנו עשרה תווים, הפעם על ידי הגדלת ערכו של משתנה האינדקס מ-101 ל-110. ערכו המדויק של המשתנה חשוב רק כאשר משתמשים בו בלולאה עצמה.

השהיית התוכנית

ניתן להשתמש בלולאת for ללא הוראות כדי להשהות את הרצת התוכנית למשך זמן נקוב:

```
for(delay=1; delay<=1000; delay++);
```

הלולאה אומנם אינה כוללת הוראות כלשהן, אך היא חוזרת על עצמה 1000 פעמים, תוך הגדלת המשתנה delay ב-1 בכל חזרה ובדיקתו לעומת התנאי. החזרה הזו גורמת להשהיה, עד שהתוכנית עוברת להוראה הבאה.

השתמש בטכניקה זו כדי להשהות את התקדמות המידע המוצג על המסך ולאפשר למשתמש די זמן כדי לקרוא את ההוראות או ההנחיות המוצגות לו. זוהי חלופה להנחיה למשתמש להקיש Enter כדי לעבור למסך הבא.

עליך לבצע מספר ניסיונות כדי לקבוע את משך ההשהיה שמתקבל מ-1000 חזרות במערכת המחשב שלך. משך ההשהיה תלוי במהירות של המחשב שברשותך. לולאה שבמחשב מהיר גורמת להשהיה בת שנייה אחת, עשויה לגרום להשהיה בת שתי שניות במחשב איטי יותר. הגדל או הקטן את מספר החזרות בתנאי הלולאה כדי להשיג את משך ההשהיה הרצוי לך.

הוראות מרובות

כדי לבצע יותר מהוראה אחת בלולאה, הקף את סדרת ההוראות בסוגריים מסולסלים. לדוגמה, להלן תוכנית שממירה 101 נתוני טמפרטורה עוקבים (מ-32 מעלות עד 132 מעלות) מפרנהייט לצלזיוס:

```
main()
{
  int temp;
  float celsius;
      puts("Fahrenheit\tCelsius\n");
      for(temp=32; temp<=132; temp++)
  {
      celsius = (5.0/9.0)*(temp-32);
      printf("%d\t\t%6.2f\n",temp, celsius);
  }
}
```



הערה

הפונקציה ()printf קובעת את הפורמט ומיישרת את הנתונים המוצגים בעמודות. שני תווי הדילוג (\t\t) מיישרים את טמפרטורות הצלזיוס עם כותרת העמודה. מציין הפורמט (%6.2) מציג את הטמפרטורות עם שתי ספרות אחרי הנקודה העשרונית, על פני שישה תווים.

בכל חזרה של הלולאה מבוצעות שתי הוראות. הערך של משתנה הבקרה קובע הן את מספר החזרות והן את הערכים שיומרו לצלזיוס. השווה את התוכנית הזו לתוכנית הבאה:

```
main()
{
      int temp, repeat;
  float celsius;
      puts("Fahrenheit\tCelsius\n");
      temp=10;
      for(repeat=1; repeat<=10;repeat++)
  {
  celsius = (5.0/9.0)*(temp-32);
      printf("%d\t\t%6.2f\n",temp, celsius);
      temp+=10;
  }
}
```

כאן קובע משתנה האינדקס רק את מספר החזרות. נתוני הטמפרטורה שיש להמיר נקבעים על ידי המשתנה temp, שגדל בדילוגים של 10 בכל חזרה - מ-10 ל-20, ל-30, וכן הלאה, עד 100.

שימוש במשתנים

גם אם בעת כתיבת התוכנית אינך יודע עדיין לכמה חזרות תזדקק, באפשרותך להשתמש בלולאת for אם תדע את מספר החזרות בעת שתריץ את התוכנית. ניתן להזין ערך למשתנה ולהשתמש בו בתנאי הלולאה. לדוגמה, התוכנית הבאה מבקשת מהמשתמש להקליד טווח של טמפרטורות להמרה - ולמעשה, את מספר הפעמים שהתוכנית תחזור על עצמה:

```
main()
{
  int temp, start, end;
  float celsius;
     printf("Enter the starting temperature: ");
     scanf("%d",&start);
     printf("Enter the ending temperature: ");
     scanf("%d",&end);
    puts("Fahrenheit\tCelsius\n");
     for(temp=start; temp<=end; temp++)
  {
  celsius = (5.0/9.0)*(temp-32);
        printf("%d\t\t%6.2f\n",temp, celsius);
  }
}
```

בתוכנית זו על המשתמש להקליד את ערך הטמפרטורה ההתחלתי והערך הסופי שברצונו להמיר. המשתנים start ו-end משמשים בפרמטר של for כדי לקבוע את הערך ההתחלתי של משתנה האינדקס וכדי לבדוק אם התנאי מתקיים. הלולאה נפסקת כאשר הערך של temp גדל מעבר לערך הטמפרטורה הסופי. לכן, אם המשתמש מקליד 20 ו-43, התוכנית ממירה מפרנהייט לצלזיוס את נתוני הטמפרטורה שבין 20 מעלות ל-43 מעלות, ועוצרת - 24 לולאות.

לולאות מקננות

כאשר מבוצעת פקודת for אחת בתוך פקודת for אחרת, הן נקראות לולאות מקננות. התוכנית חוזרת על הלולאה הפנימית בשלמותה בכל חזרה על הלולאה החיצונית. ניתן לדמיין לולאות for מקננות כבעלות שני מימדים, ולולאות for בודדות כבעלות מימד אחד.

לדוגמה, עיין בתוכנית הבאה:

```
main()
{
int row, column;
for(row=1; row <=10;row++)
  {
       for(column=1; column<=10;column++)
       printf("*");
    putchar('\n'); /*outside of second loop, inside first*/
  }
}
```

התוכנית מציגה 10 שורות של 10 כוכביות ומשתמשת בשני משתנים מטיפוס מספר שלם: row ו-column. הלולאה החיצונית מגדילה את ערכו של row מ-1 ל-10. בכל חזרה על הלולאה החיצונית, מבוצעת הלולאה הפנימית 10 פעמים, מגדילה את ערכו של column ומציגה שורה של 10 כוכביות. (שים לב ששמות המשתנים מסייעים להבהרת הלוגיקה של התוכנית). הקינון של הלולאות מודגם בתרשים 9.3. הלולאה הפנימית היא:

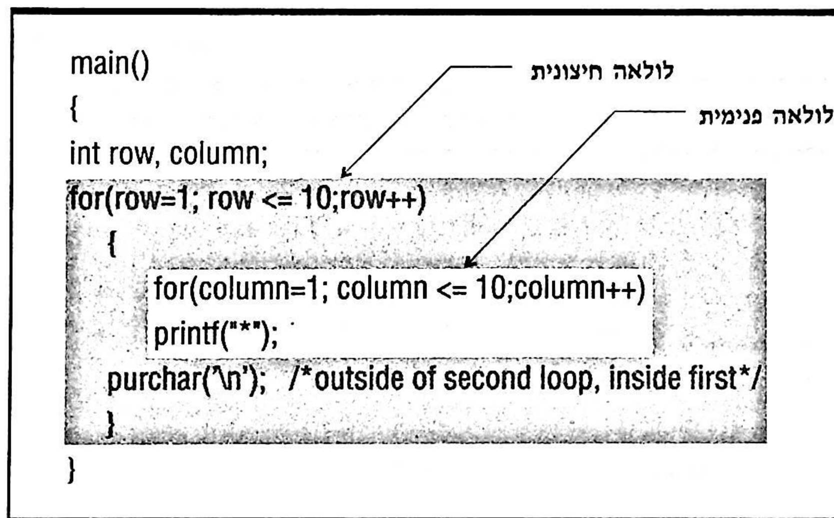
```
for(column=1; column<=10;column++)
printf("*");
```

לפיכך, מוצגות בסך הכל 100 כוכביות - 10 לולאות פנימיות (עמודות) לכל אחת מ-10 הלולאות החיצוניות (שורות). תרשים 9.4 מציג את ערכי המשתנים בכל חזרה.

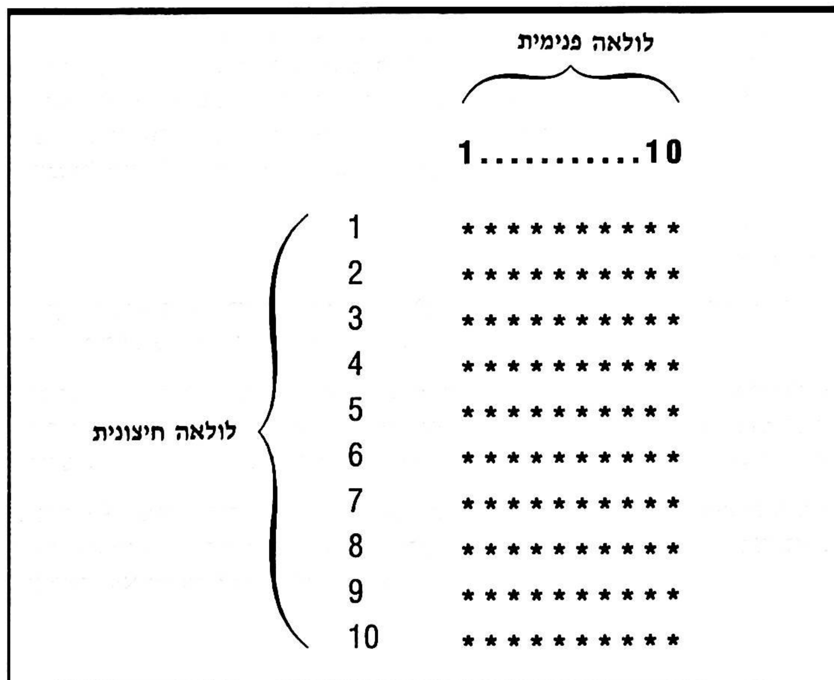
שים לב למיקומה של ההוראה ;putchar('\n'). ההוראה מופיעה מחוץ ללולאה הפנימית, אך בתוך הסוגריים המסולסלים של הלולאה החיצונית. ההוראה מבוצעת עשר פעמים, פעם אחת בכל חזרה של הלולאה החיצונית, ומחדירה פקודת שורה-חדשה בסופה של כל שורה.

תוכנית 9.1 מציגה דוגמה נוספת ללולאות מקננות. התוכנית מבצעת עשר לולאות חיצוניות ועשר לולאות פנימיות כדי ליצור את לוח-הכפל. במקום להציג כוכבית בלבד, כמו בדוגמה הקודמת, היא מציגה את מכפלת ערך השורה בערך העמודה.

תרשים 9.3
לולאה פנימית ולולאה חיצונית.



תרשים 9.4
ערכי המשתנים בכל חזרה.



תוכנית 9.1: תוכנית לוח הכפל

```
/*timestab.C/
main()
{
int row, column;
     puts("\t\tMy Handy Multiplication Table\n\n");
     for(row=1; row <=10;row++)
     {
          for(column=1; column<=10;column++)
          printf("%6d", row*column);
          putchar('\n');
     }
}
```

הפלט של timestab.c מוצג בתרשים 9.5.

תרשים 9.5
הפלט של תוכנית לוח הכפל.

My Handy Multiplication Table

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

לבסוף, עיין בתוכנית הבאה:

```
main()
{
int row, column;
for(row=1; row <=10;row++)
  {
  for(column=1; column<=row;column++)
  printf("*");
  putchar('\n');
  }
}
```

התוכנית מציגה סדרת כוכביות, כמודגם בתרשים 9.6.

תרשים 9.6
מבנה הפלט כאשר מספר החזרות על הלולאה הפנימית מבוקר על ידי הערך של הלולאה החיצונית.

```
*
**
***
****
*****
******
*******
********
*********
**********
```

לכל שורת כוכביות אורך שונה. במקרה זה, הלולאה הפנימית אינה חוזרת מספר קבוע של פעמים, אלא מספר החזרות גדל עם כל חזרה של הלולאה החיצונית - כוכבית אחת בשורה הראשונה, שתי כוכביות בשורה השנייה, שלוש כוכביות בשלישית, וכן הלאה. מספר העמודות בכל שורה זהה למספר השורה. אנו משיגים את התוצאה הזו על ידי שימוש במשתנה row בתנאי של הלולאה הפנימית.

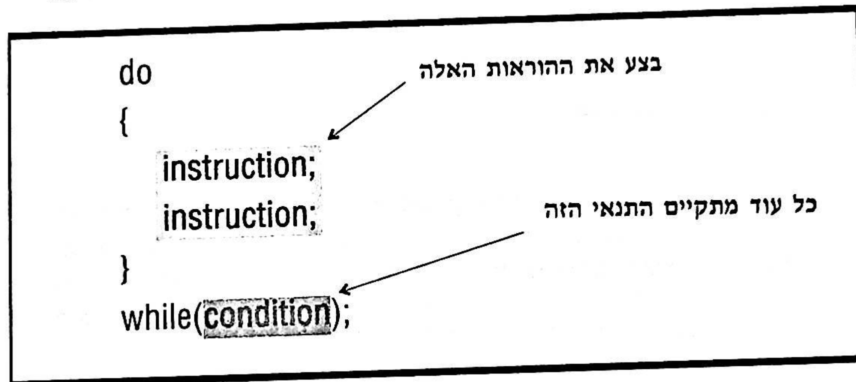
בחזרה הראשונה על הלולאה החיצונית, מבוצעת הלולאה הפנימית פעם אחת ומציגה כוכבית אחת. בחזרה השנייה על הלולאה החיצונית מבוצעת הלולאה הפנימית פעמיים, ומציגה שתי כוכביות. תהליך זה נמשך ויוצר את התבנית המוצגת בתרשים.

יש להיזהר כאשר כותבים תוכנית עם שתי רמות, או יותר, של ביטויי for. תוכנית שמבצעת 100 חזרות על לולאה חיצונית ו-100 חזרות על לולאה פנימית מבצעת, למעשה, 10,000 חזרות!

שימוש בלולאת do...while

השתמש בלולאת do...while כאשר אינך יודע מהו המספר המדויק של חזרות, אך ידוע לך שעל התוכנית לבצע את הלולאה לפחות פעם אחת. מבנה הפקודה מודגם בתרשים 9.7.

תרשים 9.7
המבנה של לולאת do...while.



ההוראות הנתונות בסוגריים מסולסלים מבוצעות שוב ושוב, כל עוד מתקיים התנאי שבביטוי ה-while. מכיוון שהתנאי אינו נבחן עד לסיום הלולאה, הלולאה תתבצע לפחות פעם אחת - אפילו במקרה שהתנאי אינו מתקיים כבר קודם לביצוע הלולאה.

בלולאת do...while עליך לוודא שהלולאה תסתיים. אם התנאי מתקיים תמיד, הלולאה תמשיך להתבצע עד שתאתחל מחדש את המחשב או תקיש על צירוף המקשים Ctrl+Break.

לעתים קרובות משתמשים במבנה של לולאת do...while כדי לחזור על תוכנית, עד שהמשתמש יחליט שאין יותר קלט:

```
main()
{
int temp;
float celsius;
char repeat;
do
{
printf("Input a temperature: ");
```

```
scanf("%d", &temp);
celsius = (5.0/9.0)*(temp-32);
printf("%d degrees F is %6.2f degrees celsius\n",temp,
celsius);
printf("Do you have another temperature?");
repeat=getchar();
putchar('\n');
}
while (repeat== 'y' || repeat=='Y');
}
```

התוכנית כולה, למעט הכרזות המשתנים, נתונה בתוך לולאת do...while אחת גדולה. הלולאה מתבצעת שוב ושוב, עד שהמשתמש מקליד תו שאינו Y או y בתגובה לבקשת הקלט. שים לב שהוראת הקלט השואלת את המשתמש אם ברצונו לחזור שוב על הלולאה היא ההוראה האחרונה בסדרה.



הערה

כדי להימנע מהתנגשות עם הקלט של ()scanf, השתמש בפקודה ()getch או ()getche במקום הפונקציה ()getchar, או לחילופין, השתמש בפקודה fflush(stdin) כדי לרוקן את החוצץ. עיין בפרק 8 למידע נוסף.

המבנה של do...while משמש גם כדי להבטיח שמוקלד הקלט המתאים. לדוגמה, נניח שכתבת תוכנית שמזינה שיעור הנחה בפורמט עשרוני. אם המשתמש מקליד ערך קטן מ-0, או ערך גדול מ-1, עליך לבקש קלט אחר. לולאת ה-do...while תיראה כך:

```
do
{
    printf("Please enter the discount: ");
    scanf("%f", &discount);
}
while(discount<0 || discount >=1);
```

תנאי ה-while משתמש באופרטור הלוגי **או** כדי לבחון אם הערך שהוקלד הוא מחוץ לתחוּם הערכים המותר. הלולאה תתבצע כל עוד המשתמש מקליד ערכים שאינם תקפים.

אין כל פסול ברוטינה הזו כל עוד המשתמש מבין, בסופו של דבר, את הכוונה - ומקליד ערך תקף. לעומת זאת, אם המשתמש אינו יודע כלל מהו מספר עשרוני, הלולאה עלולה לחזור על עצמה עד אין-קץ. פתרון אפשרי אחד הוא להוסיף מונה לאלגוריתם, כך:

```
main()
{
    int count;
    float discount;
```

```
count=0;
do
{
printf("Please enter the discount: ");
scanf("%f", &discount);
count++;
}
while((discount<=0 || discount >=1) && count <20);
if (count==20) puts("Stupid");
}
```

עתה, יש למשתמש 20 ניסיונות להקליד את הקלט המתאים! המשתנה count גדל ב-1 עם כל עיול בלתי-תקף.

כדי לבצע הוראה אחת בלבד בלולאת do...while, השמט את הסוגריים המסולסלים. התוכנית הבאה, לדוגמה, מאפשרת למשתמש להקליד תווים, עד שיקיש Y:

```
main()
{
int a;
    do
        a = getchar();
    while(a!='y' && a!='Y');
}
```

לולאות do מקננות

ניתן להציב לולאות do...while זו בתוך זו, כדי ליצור מספר רמות של חזרות. באפשרותך להשתמש בלולאה חיצונית כדי לחזור על התוכנית כולה עד שהמשתמש יחליט לחדול מכך, ובלולאות פנימיות כדי לבחון שמוקלד קלט מתאים, כפי שמודגם בתרשים 9.8. הלולאה הפנימית משמשת להזנת מספר בין 0 ל-100. כאשר מוקלד עיול תקף, הלולאה מסתיימת ומתבצעות ההוראות הנותרות בלולאה החיצונית. הלולאה החיצונית חוזרת כל עוד מתקיים התנאי הבא:

```
while (repeat== 'y' || repeat=='Y');
```

ברגע שמוקלד תו כלשהו שאינו y או Y, הלולאה החיצונית - והתוכנית כולה, מסתיימת.

תרשים 9.8
לולאות do...while מקננות.

הלולאה החיצונית חוזרת על התוכנית כולה

הלולאה הפנימית בוחנת את תקפות הקלט

```
/* nest_do.c */
main()
{
char repeat;
int temp;
float celsius;
do
{
do
{
printf("Input a valid temperature:");
scanf("%d", &temp);
}
while (temp<0 II temp > 100);
celsius = (5.0/9.0)*(temp-32);
printf("%d degree F is %6.2f degree celsius\n", temp, celsius);
printf("Do you have another temperature?");
repeat= gerchar();
putchar('\n');
}
while (repeat=='y' II repeat=='y');
}
```

שימוש בלולאת while

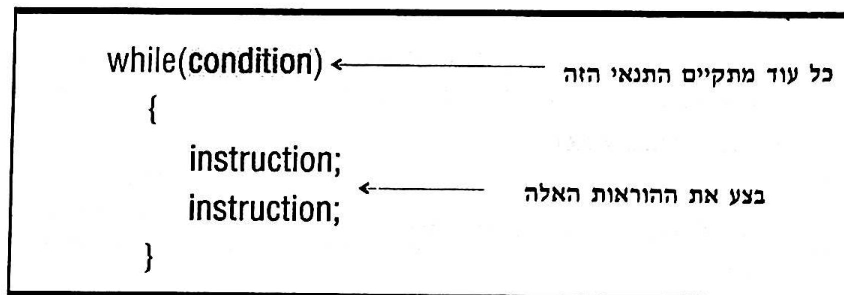
השתמש בלולאת while כאשר אינך יודע מהו מספר החזרות הדרוש, וכאשר קיימת גם אפשרות שאינך מעוניין לבצע את הלולאה כלל. מבנה הלולאה הוא כדלקמן:

```
while(condition)
instruction;
```

תרשים 9.9 מדגים את המבנה של לולאת while. כדי לכלול בלולאה מספר הוראות, השתמש במבנה הבא:

```
while (condition)
{
 instructions
}
```

תרשים 9.9
המבנה של לולאת while.



בדומה ללולאת do, ההוראות יבוצעו כל עוד מתקיים תנאי ה-while. אולם בניגוד ללולאת do, התנאי נבחן עוד קודם לביצוע הלולאה. אם התנאי אינו מתקיים, הלולאה לא תתבצע אפילו פעם אחת.

כדי להבטיח קלט מתאים, הצב את פקודת הקלט הראשונה מחוץ ללולאה. כך ניתן להמשיך ולבקש מהמשתמש לספק קלט, כל עוד הערך המוקלד אינו תקף:

```
printf("Please enter the discount: ");
scanf("%f", &discount);
while(discount<0 || discount >=1)
    {
        printf("You got it wrong ");
        scanf("%f", &discount);
    }
```

שילוב של לולאות מסוגים שונים

ניתן להציב כל שילוב של לולאות while, for ו-do...while זו בתוך זו, אם הלוגיקה של התוכנית דורשת זאת. התוכנית הבאה, לדוגמה, מציבה לולאת while מקננת בין החזרות של לולאת for, כדי להמיר 10 נתוני טמפרטורה, בין 0 ל-100, שמוזנים מהמקלדת:

```
/*mixed.C*/
main()
{
   int temp, count;
   float celsius;
   for(count=1; count<=10; count++)
     {
      printf("Enter a temperature between 1 and 100: ");
      scanf("%d",&temp);
   while (temp<0 || temp > 100)
   {
          printf("Invalid temperature, try again: ");
        scanf("%d", &temp);
 }
 celsius = (5.0/9.0)*(temp-32);
 printf("%d fahrenheit is %6.2f celsius\n",temp, celsius);
 }
}
```

לולאת for החיצונית תחזור 10 פעמים בלבד. הלולאה הפנימית תחזור כל עוד המשתמש מקליד ערך בלתי-תקף.

עיצוב התוכנית

השימוש בלולאות מוסיף תחכום לעיצוב התוכנית. יש ללמוד ולבחון את האלגוריתם בקפידה כדי לוודא שהוא פועל כהלכה.

השלב הראשון הוא להחליט באיזה מבנה להשתמש - לולאות for, do...while או while. שאל את עצמך:

- האם ידוע לי מספר החזרות הנדרש מהלולאה, או שאדע את המספר עם הרצת התוכנית?

אם התשובה חיובית, השתמש בלולאת for. אם התשובה שלילית, שאל את עצמך שאלה נוספת:

- האם ברצוני לבצע את הלולאה לפחות פעם אחת?

אם התשובה חיובית, השתמש בלולאת do...while. אם התשובה שלילית, השתמש בלולאת while.

לדוגמה, נניח שברצונך לחשב את הממוצע של סדרת מספרים. מכיוון שאינך יודע מראש כמה מספרים כוללת הסדרה, עליך להשתמש בלולאת do...while או בלולאת while - במקום בלולאת for. נניח גם שברצונך לאפשר למשתמש לעצור את התוכנית אחרי הפעלתה, בלי לבצע את הלולאה אפילו פעם אחת. לשם כך, תשתמש בלולאת while.

אבל איך תפסיק את החזרות? יש לספק למשתמש אמצעי כלשהו להורות לתוכנית שהוא סיים את הקלדת המספרים וברצונו להפסיק. דרך אחת לעשות זאת היא להציג למשתמש שאלה, בכל חזרה, אם ברצונו להקליד מספר. אולם דרך זו מחייבת את המשתמש להקליד שני עיולים בכל חזרה על הלולאה - את המספר המשמש כקלט, וכן Y או y, כדי לחזור על הלולאה פעם נוספת.

דרך טובה יותר היא להשתמש בערך קלט שיפעיל את סיום הרוטינה, כפי שמודגם בתוכנית 9.2. תוכנית זו נעצרת כאשר מוקלד מספר שלילי. הערך הראשון מוקלד לפני הלולאה, כדי שניתן יהיה מייד לעצור את התוכנית על ידי הקלדת מספר שלילי בתור הערך הראשון. עבור כל ערך חיובי שמוקלד, התוכנית מגדילה את ערכו של משתנה מונה ואוגרת את הסכום. שים לב לכך שיתר המספרים מוזנים לקראת סוף הלולאה, ממש לפני שנבחן תנאי ה-while בתחילתה של החזרה הבאה.

שים לב לתנאי:

```
if(count>0)
```

המופיע לפני חישוב הממוצע. תנאי זה חיוני כדי למנוע חילוק ב-0, שיגרום לשגיאת הרצה ברוב מערכות המחשב.

תוכנית 9.2: חישוב ממוצעים

```
/*average.C*/
main()
{
 float number, total;
 int count;
 total =0.0;
 count=0;
 printf("Enter a number, negative to end: ");
 scanf("%f",&number);
 while(number>=0)
 {
  count++;
  total+=number;
```

```
   printf("Enter a number, negative to end:");
   scanf("%f",&number);
  }
  If(count>0)
      {
      number=total/count;
 printf("Total=%.2f  Count=%d  Average=%.2f",number, count,
total);
      }
}
```

נניח לרגע שכתבת את התוכנית באופן שכל הקלט מוזן בתוך הלולאה, כך:

```
while(a>=0)
{
scanf("%f",&a);
count++;
total+=a;
}
```

כאשר יוקלד מספר שלילי כדי לעצור את התוכנית, הוא ייחשב, שלא כנדרש, לעיול תקף - ויתווסף לסך-הכל, לפני שתנאי ה-while ייבחן שנית.

כאשר משתמשים בלולאות מקננות, יש לנקוט זהירות באופן השימוש במונים ובאוגרים. לדוגמה, נניח שהחלטת לחזור על תוכנית הממוצעים, פשוט על ידי הקפתה ברמה נוספת של do...while, כמודגם בתוכנית 9.3. תוכנית זו לא תפעל כהלכה, בגלל מיקומן של ההוראות:

```
total =0;
count=0;
```

תוכנית 9.3: מיקום לא-נכון של פעולות הקצאת ערכים

```
/*ave_bad.C*/
main()
{
 char repeat;
 float number, total;
 int count;
 total =0.0;
```

```
count=0;
do
{
  printf("Enter a number, negative to end: ");
  scanf("%f",&number);
  while(number>=0)
    {
    count++;
    total+=number;
    printf("Enter a number, negative to end:");
    scanf("%f",&number);
    }
    If(count>0)
    {
  number=total/count;
  printf("Total=%.2f Count=%d Average=%.2f",number, count,
total);
  printf("Do you have another series of numbers?");
  repeat=getchar();
  putchar('\n');
   }
}
while (repeat== 'y' || repeat=='Y');
}
```

בגלל שפעולות הקצאת הערכים ממוקמות מחוץ ללולאה הראשית, הן מבוצעות פעם אחת בלבד, עם הפעלת התוכנית. בכל פעם שמתבצעת חזרה על הלולאה הראשית עבור סדרת מספרים חדשה, המונים והאוגרים שומרים על ערכיהם. התוצאה תהא ממוצע מצטבר של כל המספרים שהוקלדו. על ההוראות להופיע בתוך הלולאה החיצונית, כך:

```
do
{
total =0;
count=0;
```

מבנה זה יקצה מחדש את הערך ההתחלתי אפס למונה ולאוגר, עבור כל סדרה של מספרים. שים לב שבתוכנית מקננת לולאת while בתוך לולאת do...while.

שימוש בדגלונים

דגלון הוא אלגוריתם המסמן לתוכנית שתנאי כלשהו התקיים - בדיוק כפי שדגל המונף על הירח מסמן שהאסטרונאוטים נחתו כאן. משתנה דגלון מקבל ערך בתחילת התוכנית או בלולאה החיצונית, ובשלב מאוחר יותר מוקצה לו ערך אחר, כדי לסמן שפעולה כלשהי התרחשה או שהתוכנית הגיעה לערך מסוים.

לדוגמה, תוכנית המרת הטמפרטורות עושה שימוש בלולאת do...while כדי להזין ערכים:

```
do
{
printf("Input a valid temperature: ");
scanf("%d", &temp);
}
```

בכל חזרה מופיעה אותה ההנחיה למשתמש - בין אם זהו הקלט הראשון ובין אם המשתמש הקליד עיול בלתי-תקף. האם לא עדיף להציג הודעה שונה, שתורה למשתמש שהמספר שהקליד אינו נכון? זהו בדיוק מה שעושה תוכנית 9.4.

תוכנית 9.4: שימוש במשתנה דגלון כדי להציג הודעות שונות

```
/*flag.C*/
main()
{
 int temp;
 float celsius;
 char repeat;
 char flag;
 do
 {
   flag='n';
   do
   {
     if (flag=='n')
     printf("Input a valid temperature :");
     else
     printf("Input a valid temperature, stupid :");
     scanf("%d", &temp);
```

```
    flag='y';
   }
 while  (temp<0 || temp > 100);
 celsius = (5.0/9.0)*(temp-32);
 printf("%d degrees F is %6.2f degrees celsius\n",temp, celsius);
printf("Do you have another temperature?");
 repeat=getchar();
 putchar('\n');
 }
 while (repeat== 'y' || repeat=='Y');
}
```

משתנה דגלון, ששמו בתוכנית flag, מקבל את הערך 'n' בתחילת כל חזרה על הלולאה החיצונית. כשמופעלת הלולאה הפנימית היא בוחנת את ערכו של משתנה הדגלון, ומציגה הודעה אחת אם ערכו של המשתנה הוא 'n', והודעה אחרת אם ערכו של המשתנה שונה מ-'n'.

כשהלולאה מתבצעת בפעם הראשונה, הערך הוא 'n' ומוצגת ההודעה הראשונה. כאשר המשתמש מקליד מספר, משתנה ערכו של משתנה הדגלון ל-'y'. עם זאת, במידה שהקלט שהקליד המשתמש הוא בלתי-תקף, תחזור הלולאה הפנימית פעם נוספת. הפעם, התנאי (flag=='n') אינו מתקיים, ולכן תוצג ההודעה השנייה.

כאשר המשתמש מקליד מספר תקף, הטמפרטורה מומרת מפרנהייט לצלזיוס, והלולאה החיצונית חוזרת פעם נוספת. הלולאה החיצונית מקצה מחדש את ערכו של משתנה הדגלון, והמשתמש יכול שוב להקליד ערך תקף. בדומה למונה ולאוגר, משתנה הדגלון מקבל את ערכו המקורי בכל חזרה של הלולאה החיצונית.

משתנה הדגלון יכול לקבל כל שם וכל טיפוס נתונים, אם כי מומלץ להשתמש בטיפוסי הנתונים char או int. גם הערך המוקצה לדגלון נתון לבחירתך. בתוכנית שלנו השתמשנו בערך n עבור עיול תקף, ובערך y עבור עיול בלתי-תקף. באפשרותך לבחור בכל ערך שעולה בדעתך.

שימוש בפקודה break

שיטת הדגלון מבצעת אומנם את המטלה, אך היא מגדילה את התקורה של התוכנית - משתנה נוסף, כמה שורות הקצאת ערכים ומבנה של if...else. ניתן להימנע מהתקורה הנוספת ולהשיג את המטרה על ידי הסבת המבנה של התוכנית לשימוש בלולאות while, כמודגם בתוכנית 9.5.

תוכנית 9.5: שימוש בלולאות while ובפקודת break

```
/*breaks.C*/
main()
{
 int temp;
 float celsius;
 printf("Input a temperature or enter 555 to stop :");
 scanf("%d", &temp);
 while (temp != 555)
 {
    while ((temp<0 || temp > 100) && temp != 555)
     {
     printf("Invalid temperature, try again: ");
     scanf("%d", &temp);
     }
    if (temp==555)
          break;
 celsius = (5.0/9.0)*(temp-32);
 printf("%d degrees F is %6.2f degrees celsius\n",temp,
celsius);
 printf("Input a temperature enter 555 to stop :");
 scanf("%d", &temp);
 }
}
```

גם תוכנית זו מציגה שתי הודעות - אחת עבור עיול תקף, והודעה שונה במהלך החזרה אחרי קלט בלתי-תקף. שים לב שתוכנית זו אינה שואלת את המשתמש אם ברצונו להקליד טמפרטורה נוספת אחרי כל חזרה. במקום זאת, התוכנית מסתיימת כאשר מוקלד הערך 555.

התנאי קובע שעיול תקף הוא מספר בין 0 ל-100, או המספר 555. העיול 555 מסיים את התוכנית בגלל ההוראה:

```
if (temp==555)
break;
```

הפקודה break מסיימת את הלולאה שבתוכה היא נמצאת, בדיוק כמו שהלולאה מסתיימת כאשר תנאי while או תנאי for אינם מתקיימים.

כל תוכניות המרת הטמפרטורה שהוצגו בפרק זה הן "נכונות". ההבדל ביניהן הוא בכך שכל אחת משתמשת בגישה שונה או באלגוריתם שונה, כדי לבצע אותה מטלה. הצגת שאלה

למשתמש אם ברצונו להקליד עיול נוסף, חוסכת אומנם למשתמש את הצורך לזכור שעליו להקליד מספר שלילי או ערך מיוחד כלשהו כדי לסיים את התוכנית, אולם היא מצריכה הקשת מקש נוסף בכל חזרה. שימוש במספר מיוחד, כמו 555, חוסך את הצורך בהקשות נוספות אלה. החיסרון טמון בכך שאינך יכול להשתמש באותו מספר כעיול תקף. לדוגמה, אם כל מספר שלילי מסיים את התוכנית, לא ניתן להמיר נתוני טמפרטורה שמתחת לאפס.

המבחן ל"נכונותה" של תוכנית הוא אם היא מורצת ללא שגיאות, וממשיכה לרוץ כל עוד המשתמש רוצה בכך.

שאלות

1. מהם הקריטריונים לבחירה בלולאת for, do או while?

2. מהן הפעולות המבוצעות בפרמטר של ביטוי for?

3. מה מפסיק לולאת for?

4. מהם לולאה מקננת?

5. כיצד משתמשים בדגלון?

6. מהי מטרתה של פקודת break?

תרגילים

1. ערוך את תוכנית 8.10 (פרק 8) כך שהתוכנית תחזור על עצמה עד שלמשתמש לא יישאר קלט נוסף.

2. כתוב תוכנית שמציגה טבלת סכומי מס קנייה בשיעור 6 אחוזים, עבור מוצרים שמחירם מ-1$ עד 50$, על-פי התצורה הבאה:

עלות	סכום המס	סך-הכל
1	$.06	$1.06
2	$.12	$1.12

3. כתוב תוכנית שמזינה עשרה מספרים בין 0 ל-25.



4. כתוב תוכנית שמציגה את הגרפיקה הבאה:

```
*****
****
***
**
*
**
***
****
*****
```

5. הסבר מה לקוי בתוכנית הבאה:

```
main()
{
float row, column;
    puts("\t\tMy Handy Multiplication Table\n\n");
    for(row=1; row <=10;row++)
    {
        for(column=1; column<=10;column+)
        printf("%d", row*column);
    }
        putchar('\n');
    }
```

10

מערכים ומחרוזות

המשתנים שפגשנו עד עתה יכולים לאחסן ערך אחד בלבד בכל פעם. אם, לדוגמה, ברצונך לחשב את הממוצע של 31 נתוני טמפרטורה, ניתן להזין למשתנה 31 ערכים שונים, ערך אחד בכל פעם, ולסכמם בעזרת אוגר. עם זאת, כאשר מזינים את הערך השני, הערך הראשון אובד; כשמזינים את הערך השלישי, אובד הערך השני, וכן הלאה. כשתסיים יהיו בידך הסכום והממוצע, אך לא המספרים המקוריים.

עד עתה, אם רצית להזין 31 ערכים ולשמור אותם לשימוש בשלב מאוחר יותר בתוכנית, היית זקוק ל-31 משתנים שונים ול-31 הוראות קלט שונות.

בפרק זה תלמד כיצד לאגור ערכים במשתנה מיוחד שנקרא *מערך*. תלמד גם עוד על העבודה עם משתני מחרוזות.

מערכים

דמיין לעצמך קבוצת אנשים הממתינה בתור בכניסה לקולנוע. הדבר היחיד המשותף להם הוא עמידתם באותו תור. העובדה שאדם מסוים עומד ראשון בתור אינה מוסרת לנו מידע נוסף עליו - אין פירושה שהוא פיקח יותר, גבוה יותר או עשיר יותר מאנשים אחרים בתור.

אולם בעוד שאנשים אלה הם יחידים נפרדים, ללא סדר מסוים, הם שייכים לאותה אסופה. מכיוון שהם חלק מאסופה, ניתן לטפל בהם כקבוצה. אם הסדרן יבקש מהעומדים בתור לזוז שמאלה, כדי שלא לחסום את דוכן הפופ-קורן, התור ינוע הצידה כיחידה אחת.

מערך פועל באופן דומה לתור. מערך הוא אסופה של ערכים שניתן לטפל בהם כקבוצה. כל פריט או ערך במערך מהווה משתנה נפרד וניתן להתייחס אליו באופן עצמאי. אך מכיוון שהם אסופים לתוך מערך, ניתן להתייחס אליהם גם כקבוצה, ובדיוק כמו שהמיקום של אדם מסוים בתור אינו מגלה לנו פרטים נוספים אודות אותו אדם, כך גם מיקומו של פריט כלשהו במערך אינו קשור כלל לערך המאוחסן באותו פריט.

כבר יצרת מספר מערכים והשתמשת בהם בצורת מחרוזות. מחרוזת היא למעשה מערך של תווים בודדים. עבדת עם מערכים כקבוצות כשהצגת מחרוזות בעזרת הפקודה ()puts. אך באפשרותך גם לטפל בכל תו בנפרד.

העבודה עם מערכים מתבצעת בשלבים. ראשית, עליך להכריז על המערך כמשתנה. שנית, יש להזין ערכים לתוך המערך. לבסוף, באפשרותך להשתמש במערך כדי לבצע לוגיקה של תוכנית.

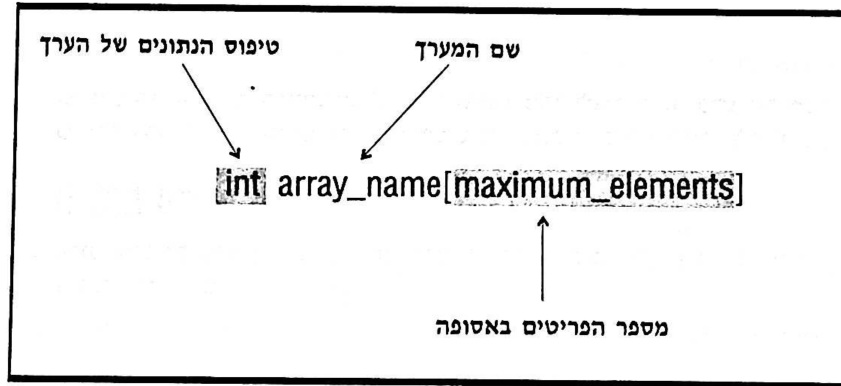
הכרזת מערך

הכרזת מערך מתבצעת על ידי קביעת טיפוס הערכים שהמערך יאחסן ואת המספר המירבי של פריטים במערך, באמצעות המבנה המודגם בתרשים 10.1. כדי ליצור מערך בן 31 מספרים שלמים, לדוגמה, השתמש בהכרזה:

```
int temps[31]
```

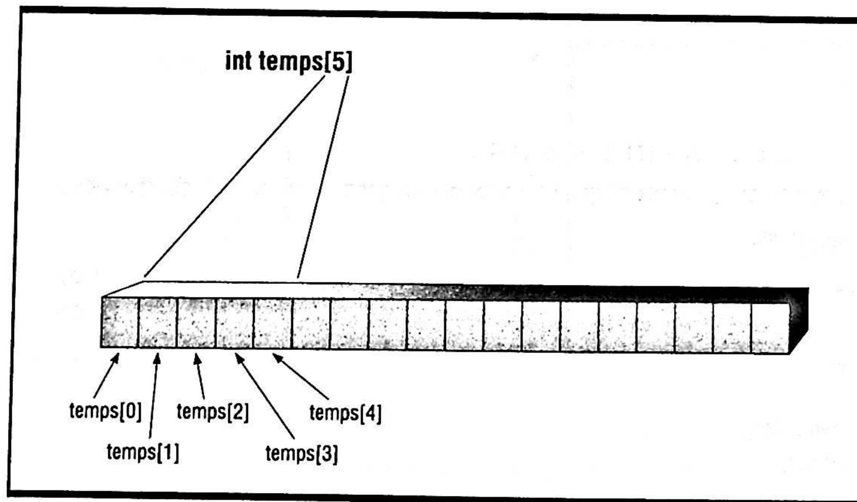
שים לב לשימוש בסוגריים מרובעים, במקום סוגריים רגילים או מסולסלים. בדומה לכל טיפוסי המשתנים, ההכרזה על מערך יכולה להתבצע לפני הפונקציה ()main, כדי שהמערך יהיה חיצוני (זמין לכל הפונקציות), או בתוך ()main או פונקציה אחרת - כדי לעשותו אוטומטי.

תרשים 10.1
המבנה של הכרזת מערך.



ההכרזה על מערך יוצרת גם את פריטי המערך, כפי שמודגם בתרשים 10.2. הפריט הראשון של המערך temps נקרא temps[0], הפריט השני temps[1], השלישי temps[2], וכן הלאה. שים לב שמספור הפריטים מתחיל מהספרה 0, ולפיכך המערך בן חמשת הפריטים המוצג בתרשים, מכיל פריטים הממוספרים מ-0 עד 4. המערך אינו כולל פריט temps[5]. המספר הנתון בסוגריים המרובעים נקרא subscript (מציין/מספר סידורי תחתי), ופריטי המערך מכונים לפיכך "temps sub 0", "temps sub 1", וכן הלאה.

תרשים 10.2
הכרזת מערך יוצרת גם את פריטי המערך.



המספר הסידורי הגבוה ביותר יהיה תמיד קטן ב-1 ממספר הפריטים המוכרזים במערך. אם הכרזת על מערך בן 10 פריטים, עליך לוודא שלא תשתמש במספר סידורי גבוה מ-9. שגיאה מסוג זה לא תתגלה על ידי הקומפיילר בתהליך הקימפול, אולם תגרום לשגיאת הרצה או לתוצאות משונות.

הכרזה על מחרוזת היא, למעשה, הכרזת מערך של משתנים מטיפוס char:

```
char name[5];
```

אולם בעוד שבעזרת הפונקציות gets()‎ ו-puts()‎ ניתן ליצור קלט ופלט של מערך שלם כיחידה אחת, טיפוסי נתונים אחרים מחייבים ביצוע הפעולות בנפרד על כל פריט במערך.

הכנסת ערכים למערך

מרגע שהכרזת על מערך, באפשרותך להכניס לתוכו ערכים. ניתן להקצות לפריטי המערך ערכים התחלתיים בעת ההכרזה:

```
int temps[5] = {45, 56, 12, 98, 12};
```

הכרזה זו יוצרת מערך בן חמישה פריטים עם הערכים הבאים:

```
temps[0] 45
temps[1] 56
temps[2] 12
temps[3] 98
temps[4] 12
```

כדי לאתחל מערך מסוג זה, הקלד את ההוראה לפני main()‎, או בתוך main()‎ או פונקציה אחרת כמשתנה סטטי:

```
int temps[5] = {45, 56, 12, 98, 12};
main()
{
static float prices[3] = {23.45, 34.56, 12.34};
```

באפשרותך גם להכניס ערכים בתוך main()‎ או בתוך פונקציה אחרת, על ידי הקצאה:

```
temps[0]=45;
```

עם זאת, כאשר ברצונך להקצות ערך לכל פריט במערך, נוח ומהיר יותר להשתמש בלולאה. אם ידוע לך מספר הפריטים שיש להקצות להם ערכים, השתמש בלולאת for:

```
main()
{
int temps[31];
int index;
for(index=0; index <31; index++)
      {
      printf("Enter temperature #%d: ", index);
      scanf("%d",&temps[index]);
      }
}
```

המשתנה index משמש כאן לקביעת מספר החזרות. בדוגמה זו, הלולאה חוזרת 31 פעמים, פעם אחת עבור כל פריט במערך. בכל חזרה של הלולאה מוצגת ההנחיה:

```
Enter temperature #
```

ולאחריה המספר הסידורי של פריט הקלט.

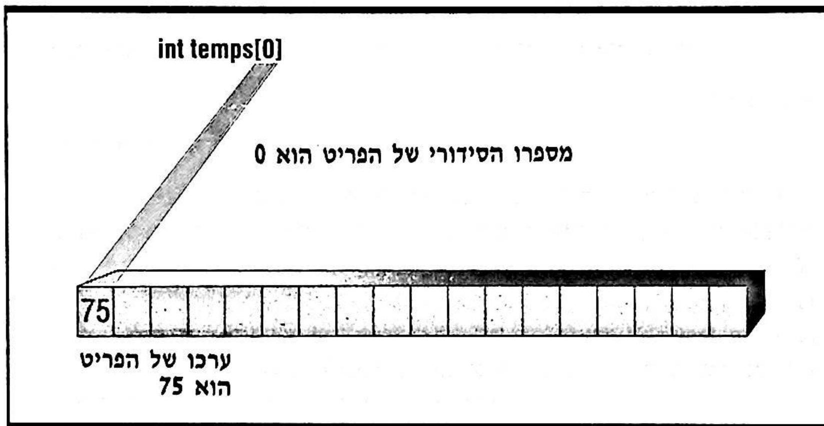
שים לב ש-31 החזרות מבוצעות על ידי הגדלת ערכו של המשתנה מ-0 ל-30, ולא מ-1 ל-31. כך ניתן להשתמש במשתנה index גם כמספר הסידורי המציין את מספרי הפריטים במערך, שממוספרים מ-0 עד 30. כאשר ערכו של המשתנה index הוא 0, הפריט temps[index] הוא למעשה temps[0]. לפיכך, הזנת ערך לתוך temps[index] באמצעות הפונקציה ()scanf, מחדירה ערך לתוך הפריט הראשון במערך, temps[0]. מכיוון שערכו של index גדל עם כל חזרה על לולאת ה-for, הפונקציה ()scanf מזינה ערך לפריט אחר של המערך בכל חזרה. טבלה 10.1 מדגימה את התהליך.

טבלה 10.1
שימוש בחזרות כדי להגדיל את מספרו הסידורי של המערך.

מספר החזרה	ערך האינדקס	הפריט המקבל את הקלט
1	0	temps(0)
2	1	temps(1)
3	2	temps(2)
4	3	temps(3)
5	4	temps(4)
6	5	temps(5)
.	.	.
.	.	.
.	.	.
26	25	temps(25)
27	26	temps(26)
28	27	temps(27)
29	28	temps(28)
30	29	temps(29)
31	30	temps(30)

חשוב להבחין בין ערכו של משתנה האינדקס בתוך הסוגריים המרובעים, לבין ערכו של פריט המערך עצמו. משתנה האינדקס מציין רק את מיקומו של הפריט במערך. הוא אינו קשור כלל לערך המאוחסן בפריט עצמו. עיין בתרשים 10.3. המספר הסידורי של הפריט הוא 0. במשתנה עצמו, temps[0], מאוחסן הערך 75.

תרשים 10.3
הבחן בין המספר הסידורי לבין הערך של פריט המערך.



כמו בדוגמה של התור הממתין בכניסה לקולנוע, המספר הסידורי אינו קשור לערכו של המשתנה. ערכו של המשתנה הבא, temps[1] יכול להיות גבוה יותר מערכו של temps[0], נמוך ממנו או זהה לו.

כמו כל משתנה, גם פריט מערך שלא הוקצה לו ערך יכיל ערך בלתי ידוע. ניתן לאתחל במהירות מערך של מספרים שלמים לאפס בעזרת הלולאה הבאה:

```
for(index=0; index <31; index++)
    temps[index]=0;
```

בעבודה עם מערך, באפשרותך להזין כל מספר של פריטים על ידי שינוי ההכרזה ותנאי הלולאה.

טיפול במערכים

ניתן להשתמש בפריט מערך בכל מקום בו משתמשים במשתנה - בהוראה, בפקודות קלט או פלט, או בביטויים. ההתייחסות לפריט מערך היא תמיד באמצעות המספר הסידורי שלו.

הערכים המאוחסנים במערך זמינים לתוכנית בכל עת. תוכנית 10.1, לדוגמה, עושה שימוש במערך לביצוע שתי מטלות שונות. תחילה מחושב הממוצע של ערכי המערך, ולאחר מכן משמש המערך להדפסת טבלת המרה. בשתי הפעמים, הגישה למערך מתבצעת באמצעות לולאת for, שמגדילה את משתנה האינדקס מ-0 ועד לפריט המערך האחרון.

תוכנית 10.1: שימוש במערך לביצוע שתי מטלות

```
/*array1.c*/
main()
{
    int temps[31];
    int index, total;
```

```
    float average, celsius;
    total=0.0;
    /*load the array with values */
    for(index=0; index < 31; index++)
    {
         printf("Enter temperature #%d:", index);
         scanf("%d",&temps[index]);
    }

    /* read the values to average */
    for(index=0; index < 31; index++)
         total+=temps[index];
    average=total/31.0;
    printf("average is: %f\n\n", average);
    puts("Fahrenheit\tCelsius\n");

    /* read the values to convert */
    for(index=0; index < 31; index++)
    {
          celsius = (5.0/9.0)*(temps[index]-32);
          printf("%d\t\t%6.2f\n",temps[index], celsius);
    }
}
```

עם זאת, התוכנית מניחה שהמשתמש יקליד ערכים לתוך כל 31 פריטי המערך. אם יוקלדו נתוני הטמפרטורה לחודשים נובמבר או פברואר, למשל, הרוטינה לא תציג תוצאות מדויקות. למעשה, אין הכרח להשתמש בכל פריטי המערך, כל עוד הלוגיקה של התוכנית מתחשבת בכך.

בתוכנית 10.2 ניתן להשתמש לתיעוד של עד 31 נתוני טמפרטורה. לולאות ה-for שבתוכנית 10.1 הוחלפו בלולאות do...while, וההוראה:

```
index=0;
```

מופיעה לפני כל לולאה כדי לאתחל את החזרות לפריט הראשון במערך. כאשר משתמשים בלולאת for אין צורך בהוראה נפרדת, מכיוון שהערך מאותחל בפרמטר הלולאה.

תוכנית 10.2: שימוש בלולאת do...while לטעינת מערך

```
/*array2.c*/
main()
{
     int temps[31];
     int index, total;
     float average, celsius, count;
     total=0.0;
     /*load the array with values */
     index=0;
     do
     {
          printf("Enter temperature #%d -- 555 to quit:", index);
          scanf("%d",&temps[index]);
          index++;
     }
     while(index< 31 && temps[index-1] != 555);
     /*average the array*/
     index=0;
     do
     {
          total+=temps[index];
          index++;
     }
     while(index< 31 && temps[index] != 555);
     count=index;
     average=total/count;
     printf("average is: %f\n\n", average);
     puts("Fahrenheit\tCelsius\n");

     /* convert the values*/
     index=0;
     do
     {
          celsius = (5.0/9.0)*(temps[index]-32);
          printf("%d\t\t%6.2f\n",temps[index], celsius);
          index++;
     }
     while(index< 31 && temps[index] != 555);
}
```

הערכים מוזנים לתוך המערך עד שיוקצו כל 31 הפריטים, או עד שיוקלד ערך הדגלון 555, באמצעות התנאי:

```
while(index< 31 && temps[index-1] != 555);
```

שים לב שערכו של משתנה האינדקס הופחת ב-1, מכיוון שהמשתנה גדל אחרי הזנת הטמפרטורה. במקרה שהקלט כולל פחות מ-31 נתוני טמפרטורה, הערך 555, המוקלד כדי להפסיק את החזרות, מוחדר לתוך פריט המערך שאחרי נתון הטמפרטורה האחרון.

הערך 555 משמש כדגלון. הלוגיקה של התוכנית נעזרת בדגלון כדי להחליט אם המשתמש סיים להקליד את נתוני הטמפרטורה לפני סוף המערך.

כאשר מחושב הממוצע של פריטי המערך, לולאת ה-do...while מסכמת את הערכים עד סופו של המערך או עד לערך 555. המשתנה index כולל אומנם את מספרם של נתוני הטמפרטורה שהוקלדו, אך כדי לשמש כמספר סידורי במערך, עליו להיות משתנה מטיפוס מספר שלם. כדי לקבל ערך מטיפוס float כמנה, הערכים המחולקים חייבים להיות מטיפוס float. לכן, התוכנית כוללת הגדרה למשתנה מטיפוס float, ששמו count, מקצה לו את מספר נתוני הטמפרטורה שהוקלדו, ומשתמשת בו לביצוע הפעולה המתמטית:

```
count=index;

average=total/count;
```

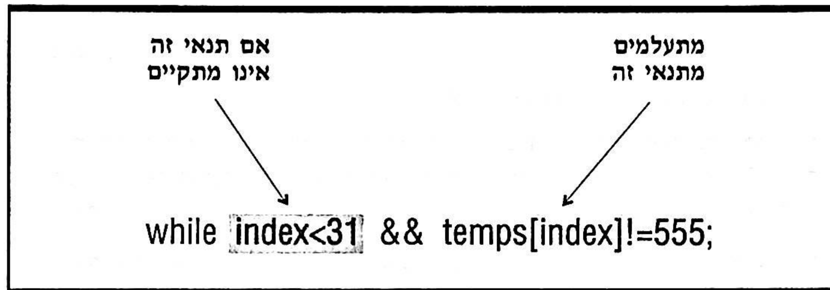
מכיוון שהשורה האחרונה בלולאה מגדילה את המשתנה index, ערכו של index יהיה, למעשה, גדול ב-1 ממספרו הסידורי של הפריט האחרון במערך שמכיל נתון טמפרטורה תקף. אם המשתמש מזין ערכים לתוך פריטים 0 עד 4, לדוגמה, ערכו של index יהיה 5. עם זאת, ערך זה מציין גם את מספר הפריטים שהוקלדו - 0 עד 4 הם 5 פריטים - ולפיכך ניתן להשתמש בו כמונה לצורך חישוב הממוצע.

אם נבחן בקפידה את הלוגיקה של התוכנית, נגלה בעיה פוטנציאלית בתנאי השני והשלישי של לולאת ה-while. אם אכן מזינים את כל 31 נתוני הטמפרטורה (0 עד 30), המספר הסידורי גדל ל-31 ומופיע בביטוי התנאי כ-temps[31], כלומר - גדול מהמספר הסידורי המירבי המותר. עם זאת, לא תוצג הודעת שגיאה בגלל סדר הופעת התנאים בביטוי:

```
while(index< 31 && temps[index] != 555);
```

כדי לקצר את משך הרצת התוכנית, מרבית הקומפיילרים מפסיקים את ההערכה של ביטוי הכולל אופרטור **וגם** ברגע שמתגלה התנאי הראשון שאינו מתקיים (כמודגם בתרשים 10.4). מכיוון שאופרטור **וגם** פירושו שעל כל התנאים בביטוי להתקיים, אין כל צורך להמשיך ולבחון את הביטוי אחרי התנאי השקרי הראשון.

תרשים 10.4
הערכת ביטוי מקוצרת.



לאחר שהוקלדו 31 פריטי מערך, התנאי הראשון (index< 31) כבר אינו מתקיים. במצב זה התוכנית אינה טורחת כלל לבחון את התנאי השני, ולכן אינה מגלה שהמספר הסידורי חרג מהטווח המותר.

ניתן לבדוק זאת בקלות על ידי שינוי סדר הופעת התנאים, כך:

```
while(temps[index] != 555 && index< 31);
```

במצב זה, אם נזין 31 ערכים ונריץ את התוכנית - תתקבל שגיאת הרצה, מכיוון שהמחשב יגלה פריט מערך שמספרו הסידורי חורג מהטווח המותר.

ייתכן שישנם קומפיילרים מסוימים שאינם משתמשים בקיצור-הדרך שתיארנו. בעבודה עם קומפיילרים כאלה תתגלה שגיאת הרצה בתוכנית. הפתרון הוא להכריז על מערך בן 32 פריטים, אך להשתמש ב-31 פריטים בלבד.

בדיקת מערך

על ידי שימוש בחזרות ניתן לסרוק את פריטי המערך, כדי להשוות בין הפריטים או לאתר ערך מסוים כלשהו. עיין ברוטינה הבאה:

```
/*highlow*/
high=temps[0];
low=temps[0];
index=1;
while(index< 31 && temps[index] != 555)
     {
     if(temps[index]>high)
        high=temps[index];
     if(temps[index]<low)
        low=temps[index];
```

```
        index++;
        }
    printf("low is %d\n",low);
    printf("high is %d\n",high);
    }
```

בהנחה שהכרזנו על המשתנים low ו-high כמשתנים מטיפוס מספר שלם, הרוטינה בודקת את הערכים של כל הפריטים במערך, כדי למצוא את הערך הנמוך ביותר ואת הערך הגבוה ביותר. המפתח הוא להקצות, בהתחלה, את הערך של הפריט הראשון במערך הן למשתנה high והן למשתנה low. אם בוחנים פריט אחד בלבד, כמובן שערכו יהיה גם הגבוה ביותר וגם הנמוך ביותר. פעולה זו "זורעת" את המשתנים, שערכיהם מושווים אז לערכים של יתר פריטי המערך. אם מתגלה במערך ערך גבוה מערכו הנוכחי של המשתנה high, הוא הופך להיות הערך הגבוה החדש. אם הערך נמוך מהערך הנוכחי של המשתנה low, הוא הופך לערך הנמוך החדש. תרשים 10.5 מציג את המחזורים הראשונים בלולאה.

תרשים 10.5
ערכי המשתנים במהלך החזרות.

ההוראה שמשנה את הערך		הערך של המשתנה Low	הערך של המשתנה High	ערך הפריט	פריט המערך
low=temps[0]	high=temps[0]	65	65	65	temps[0]
	high=temps[1]	65	95	95	temps[1]
		65	95	75	temps[2]
low=temps[3]		34	95	34	temps[3]

חיפוש במערך

לעתים קרובות עלינו לבצע חיפוש במערך כדי לאתר ערך מסוים. החיפוש כרוך בבדיקה של כל אחד מהפריטים כדי לראות אם הוא זהה לערך המסוים אותו מחפשים. אם החיפוש נועד רק לגלות אם הערך המסוים נמצא במערך, ניתן להפסיקו ברגע שמתגלה הפריט הראשון שערכו זהה לערך המבוקש. כאשר מבצעים חיפוש במערך בן אלף פריטים ומגלים את הערך המבוקש בפריט הראשון, אין צורך להמשיך ולחפש ב-999 הפריטים הנותרים.

להלן רוטינה המשמשת למציאת ערך במערך הטמפרטורות שלנו:

```
/* found.c*/
printf("Enter the value to search for: ");
scanf("%d", &num);
index=0;
found=0;
while(!found && index< 31 && temps[index]!=555)
{
      if(temps[index]==num)
         found=1;
        else
         index++;
}
  if(!found)
      puts("That number is not in the list");
 else
printf("The number is in position %d\n", index);
```

הרוטינה מניחה שהכרזנו על משתנה מטיפוס מספר שלם ששמו found. המשתנה משמש כדגלון; הוא מציין שנמצא ערך תואם ליעד החיפוש, ואין צורך להמשיך ולחפש ביתר פריטי המערך.

אנו משתמשים במשתנה found כאופרטור יחידני. הביטוי:

```
while(!found)
```

פירושו "כאשר לא נמצא (found)". הביטוי הזה מתקיים כאשר ערכו של found הוא אפס. מכיוון שהקצנו ל-found ערך התחלתי של 0 מחוץ ללולאה, התנאי מתקיים עבור החזרה הראשונה, והתוכנית בודקת את הפריט הראשון במערך (שמספרו הסידורי 0) למציאת התאמה.

כאשר נמצאת התאמה, תנאי ה-if מקצה ל-found את הערך 1. פירוש הדבר שהתנאי "כאשר לא נמצא (found)" חדל להתקיים, והלולאה מסתיימת. הערך הסופי של המשתנה index יכיל את מספרו הסידורי של הפריט המכיל את הערך התואם.

תנאי ה-if הסופי מדפיס הודעה מתאימה, בהתאם לתוצאות החיפוש.

שים לב שהוראת ה-while כוללת שלושה תנאים הקשורים ביניהם באמצעות האופרטור הלוגי **וגם**. הלולאה תחזור על עצמה כל עוד לא נמצא ערך תואם, וגם החיפוש עוד לא הגיע לסוף המערך, וגם לא אותר הערך 555.

העברת מערך

ניתן להעביר מערך שלם מפונקציה אחת לפונקציה אחרת, אם כי שפת C ושפת ++C מטפלות במערך מועבר באופן שונה לחלוטין מאופן הטיפול בטיפוסים אחרים של משתנים.

כאשר מעבירים משתנה שאינו מערך, C יוצרת עותק של הנתונים ומציבה אותו בעמדת זיכרון שמקושרת עם המשתנה הקולט. קיימים שני עותקים של הנתונים, ושינוי ערכו של המשתנה בפונקציה הקולטת אינו משנה את ערכו של המשתנה המקורי.

כשמעבירים מערך, מעבירים למעשה את כתובתו של המערך. C אינה יוצרת עותק של המערך כולו, אלא רק מקצה אותו אזור זיכרון, ובכך גם את הנתונים המאוחסנים בו, לשם שונה של מערך. תוכנית 10.3 מדגימה כיצד מעבירים מערך כארגומנט. שים לב שארגומנט הקריאה משתמש בשמו של המערך בלבד, ללא מספר סידורי:

```
convert(temps)
```

תוכנית 10.3: העברת מערך

```
/*arr_pass.c*/
#define COUNT 31
main()
{
     int temps[COUNT];
     int index;
     float celsius;
     /*load the array with values */
     for(index=0; index < COUNT; index++)
     {
          printf("Enter temperature #%d:", index);
          scanf("%d",&temps[index]);
     }
     /* pass array to convert */
     convert(temps);
}

     /*convert function*/
     convert(heat)
     int heat[];
     {
          int index;
          float celsius;
          for(index=0; index <COUNT; index++)
          {
            celsius = (5.0/9.0)*(heat[index]-32);
            printf("%d\t\t%6.2f\n",heat[index], celsius);
          }
     }
```

העברת מערכים בתקן K&R C

ההגדרה המקורית של K&R C אינה מאפשרת העברת מערך על ידי שימוש בשמו, כפי שעשינו בתוכנית 10.3. כדי להעביר מערך ב-K&R C, עליך להשתמש במשתנה מצביע כפי שנלמד בפרק 11. לעומת זאת, הקומפיילרים של ANSI C ושל C++ מאפשרים להעביר מערך באופן שמודגם בפרק זה.

מכיוון שאנו מעבירים למעשה את כתובתו של המערך, אין צורך לציין את מספר הפריטים בארגומנט הקולט; די להשתמש בסוגריים מרובעים כדי לציין שזהו מערך:

```
convert(heat)
int heat[];
```

הערות C++

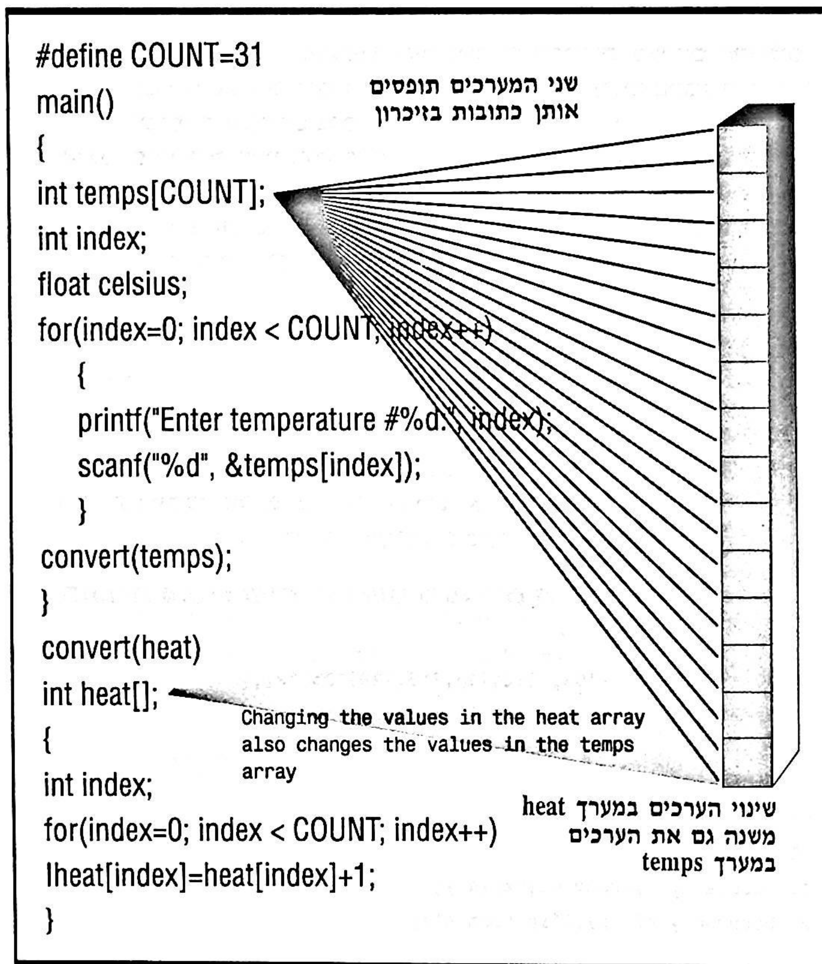
זכור, בשפת C++ ניתן להכריז על המערך בתוך הארגומנט, כך:
convert(heat[])

העברת מערך באמצעות כתובתו, במקום לשכפל אותו, חוסכת זיכרון, אולם עשויות להתלוות לה תופעות לוואי. שינוי של פריט מערך בפונקציה הקולטת יגרום גם לשינוי הערך במערך המועבר. אם הפונקציה convert() מקצה ערכים חדשים למערך heat, הערכים החדשים ייושמו גם במערך temps, כפי שמודגם בתרשים 10.6. בתוכנית שלנו הפונקציה convert() מוסיפה 1 לכל אחד מפריטי המערך heat. המערך heat תופש אותה כתובת זיכרון כמו המערך temps, ולכן גדל ב-1 גם כל אחד מהפריטים במערך temps.

תרגיל במערכים

כדוגמה לשימוש במערכים, הבה נניח שאתה מנהל מרכז ועידות בבית-מלון. לרשותך עומדים עשרה חדרי ישיבות, כשלכל אחד מהם תכולת אנשים מותרת משלו. אתה זקוק לתוכנית שמבצעת את שלוש הפעולות הבאות:

- מציגה תרשים המראה את מספרו של כל חדר ישיבות ואת תכולתו המירבית.
- מציגה את מגבלת התכולה של חדר ישיבות מסוים.
- מציגה רשימת חדרים בעלי תכולה מירבית מסוימת.



תרשים 10.6
שינוי ערכים של מערך בפונקציה.

ישנן כמה דרכים לבנות תוכנית כזו. באפשרותך להשתמש במערך בן 20 פריטים, שמכיל הן את מספרי החדרים והן את ערכי התכולה, כך:

```
int room[20]={102, 12, 107, 43.....
```

המערך מאותחל כך שאחרי כל מספר חדר מופיעה תכולתו. חדר 102 מכיל 12 אנשים, חדר 107 יכול להכיל 43 אנשים, וכן הלאה. אם מאתרים את מספר החדר במערך, ידוע שמגבלת התכולה שלו מאוחסנת בפריט הבא אחריו.

אפשרות נוספת היא להשתמש במערך דו-מימדי. נעיף מבט במערכים דו-מימדיים בשלב מאוחר יותר, כשנדון במחרוזות.

האפשרות השלישית היא להשתמש בשני מערכים מקבילים. מערכים מקבילים הם שני מערכים נפרדים שיכולים להתייחס זה לזה. ניתן לדמיין מערכים מקבילים כשני טורים של בוגרי אוניברסיטה העומדים בטקס הענקת התעודות. כאשר שני הטורים מתקרבים לכניסה לאולם, סטודנטים משני הטורים נעמדים זה לצד זה. האדם הראשון בטור הימני יצעד פנימה יחד עם האדם הראשון בטור השמאלי, וכן הלאה. אם הבוגרים התייצבו בטורים בצורה אקראית, אין קשר בין העומדים בטור השמאלי לבין העומדים בטור הימני. אולם ניתן להתאים את הטורים כך שייווצרו זוגות מסוימים כאשר הטורים נפגשים.

כאשר משתמשים במערכים מקבילים, יוצרים שני מערכים נפרדים שסדר הפריטים בהם מתואם. הפריט הראשון במערך אחד מתייחס באופן כלשהו לפריט הראשון במערך השני, וכן הלאה.

תוכנית 10.4 מדגימה פתרון לבעיית ניהול החדרים שהצגנו, בעזרת מערכים מקבילים. המערך room מכיל רשימה של עשרת מספרי החדרים הזמינים. המערך max מכיל את מגבלות התכולה של אותם עשרה חדרים. אם אנו מאתרים את מספר החדר המבוקש בפריט 5 במערך room, ידוע לנו שתכולתו מופיעה בפריט 5 של המערך max.

תוכנית 10.4: שימוש בשני מערכים מקבילים

```
/*rooms.c*/
int room[10]={102, 107, 109, 112,115,116,123,125,127,130};
int max[10]={12,43,23,12,20,15,16,23,12,15};
main()
{
int index, choice, num, rooms, flag, found;
rooms=10;
puts("1. Occupancy by room number\n");
puts("2. Occupancy of specific room\n");
puts("3. Find rooms with occupancy\n");
printf("Enter choice 1 to 3: ");
scanf("%d", &choice);
if(choice==1)
{
     for(index=0;index <rooms; index++)
     printf("Room %d  %d maximum\n",room[index],max[index]);
}
if(choice==2)
{
     printf("Enter room number: ");
     scanf("%d", &num);
     index=1;
     found=0;
```

```
        while(!found && index<rooms)
            if(room[index]==num)
                found=1;
            else
                index++;

        if(!found)
             puts("That room number is not for class use\n");
          else
    printf("Room %d holds %d persons\n", room[index],max[index]);
    }
    if(choice==3)
    {
       flag=0;
           printf("Enter the minimum number of persons: ");
         scanf("%d", &num);
         for(index=0; index<rooms; index++)
           if(max[index]>=num)
               {
               flag=1;
printf("Room %d holds %d persons\n",room[index],max[index]);

    }
        if(flag==0)
            puts("There are no rooms that large\n");
    }
    }
```

התוכנית מקצה את המספר הסידורי המירבי של המערכים למשתנה rooms, ומציגה תפריט של אפשרויות. הבחירה שלך קובעת איזו פונקציה תתבצע, בסדרה של שלוש הוראות if. באותה מידה, ניתן היה גם לכתוב את התוכנית עם הוראת switch או עם הוראות if...else מקננות.

רוטינת ה-if הראשונה משתמשת בלולאת for כדי להדפיס טבלה של מספרי החדרים ומגבלות התכולה. יש צורך במשתנה מספר סידורי אחד בלבד, והוא משמש כמספר סידורי הן עבור המערך room והן עבור המערך max.

רוטינת ה-if השנייה מזינה מספר חדר, וסורקת את המערך למציאת התאמה.

הרוטינה השלישית מזינה ערך תכולה ומציגה רשימה של כל החדרים שיכולים להכיל את הערך הזה. במקום לעצור כאשר היא מוצאת את ההתאמה הראשונה, הרוטינה סורקת את כל המערך כדי להדפיס טבלה של כל החדרים המתאימים למטרה. הרוטינה משתמשת במשתנה ששמו flag כדי לציין שלא אותרו חדרים מתאימים.

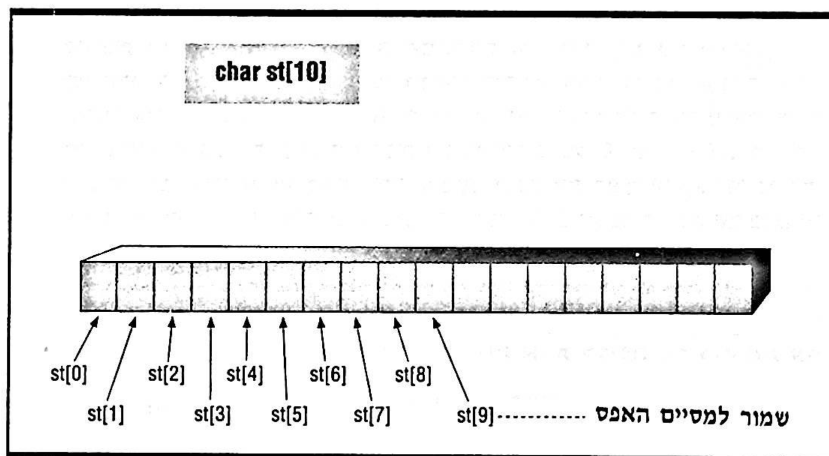
מחרוזות

מחרוזת היא מערך של תווים. כאשר מכריזים על מחרוזת יש להעניק לה שם ולציין את המספר המירבי של תווים שביכולתה לאחסן. הפריט האחרון במערך, עם זאת, שמור עבור מסיים האפס (\0), ולכן יש להכריז על המערך כבעל פריט אחד יותר מהמספר המירבי של תווים בפועל.

שפת C ושפת ++C מאפשרות לנו ליצור קלט ופלט של מערך תווים כיחידה אחת - מחרוזת. אולם כל תו הוא למעשה פריט נפרד במערך, כפי שמודגם בתרשים 10.7. לדוגמה, ניתן להזין מחרוזת ולהציג את תוויה בנפרד, כך:

```
main()
{
char name[20];
int index;
printf("Enter your name: ");
scanf("%s", name)
for(index=0, index<20;index++)
      printf("%c\n",name[index]);
}
```

אם אין מזינים את כל 20 התווים, יכילו הפריטים שאחרי מסיים האפס ערכים בלתי ידועים.



תרשים 10.7
כל תו במחרוזת הוא למעשה פריט נפרד במערך.

גם אילו לא היתה קיימת הפונקציה ()gets בשפת C, ניתן היה להזין מחרוזת על ידי הקלדת סדרת תווים והקצאתם לפריטי המערך הנפרדים, כפי שמודגם בתוכנית 10.5.

תוכנית 10.5: חיקוי פעולתה של הפונקציה ()gets

```
/*getstr.c*/
main()
{
     char name[10], letter;
     int index;
     index=0;
     puts("Enter a name, press Enter when done\n");
     do
     {
          letter=getchar();
          name[index]=letter;
          index++;
     }
     while(letter !='\r' && index<9);
     name[index]='\0';
     putchar('\n');
     puts(name);
}
```

למרבית המזל, C מכירה בעובדה שמחרוזות הן פשוט סוג של מערכים, בכך שהיא מאפשרת לנו ליצור קלט ופלט של המערך כיחידה אחת. למעט העובדה הזאת, C לא תוכננה עם שפע של פונקציות מיוחדות אחרות לטיפול במחרוזות. אך מכיוון שמחרוזות הן כה שימושיות בתכנות, כוללים מרבית הקומפיילרים של C ושל ++C פונקציות מיוחדות לעבודה עם מחרוזות. יש באפשרותך, אומנם, לבצע את הפעולות על ידי כתיבת הוראות משלך, אך הפונקציות המובנות יעילות יותר. טבלה 10.2 מציגה רשימה של כמה מפונקציות המחרוזות האופייניות שניתן למצוא בספריות ++C.

טבלה 10.2
פונקציות לטיפול במחרוזות.

פונקציה	תיאור
strcat	**מוסיפה תו ממחרוזת אחת לסופה של מחרוזת אחרת.**
strchr	**מחזירה את מיקומו במחרוזת תו נתון.**
strcmp	**משווה בין שתי מחרוזות.**
strcmpi	**משווה בין שתי מחרוזות; ללא הבחנה בין אותיות גדולות לקטנות.**
strcpy	**מעתיקה מחרוזת אחת, או ערך מחרוזת מילולי, אל מחרוזת אחרת.**
strcspn	**מחזירה את מיקומו של תו במחרוזת מתוך סדרת תווים נקובה.**
strdup	**משכפלת מחרוזת.**
strlen	**מחשבת את אורכה של המחרוזת.**
strlwr	**ממירה מחרוזת לאותיות קטנות.**
strncat	**מוסיפה תווים נקובים ממחרוזת אחת לאחרת.**
strncmp	**משווה בין תווים נקובים בשתי מחרוזות.**
strncpy	**מעתיקה תווים נקובים ממחרוזת אחת לאחרת.**
strnset	**משנה תווים נקובים במחרוזת לתווים אחרים.**
strrev	**הופכת את התווים במחרוזת.**
strstr	**מאתרת מחרוזת אחת בתוך מחרוזת אחרת.**

כדוגמה, נבחן כמה פונקציות לטיפול במחרוזות ונראה כיצד ניתן לבצע אותן לבד.

השוואה בין שתי מחרוזות

בשפת C ובשפת ++C לא ניתן להשוות באופן ישיר בין הערכים של שתי מחרוזות על ידי תנאי כגון:

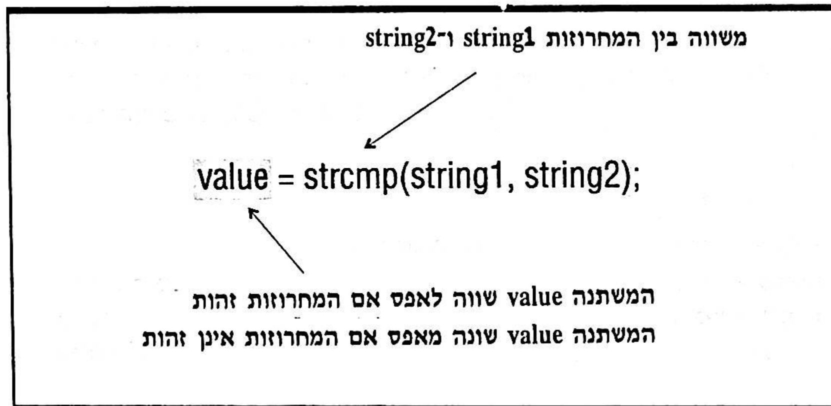
```
if(string1==string2)
```

עם זאת, מרבית הספריות כוללות את הפונקציה ()strcmp, שמחזירה ערך אפס אם שתי המחרוזות שוות, או ערך שאינו אפס אם המחרוזות אינן זהות. המבנה של הפונקציה ()strcmp מודגם בתרשים 10.8, ומשמש בקטע התוכנית הבא:

```
if(strcmp(name1,name2)==0)
    puts("The names are the same");
    else
    puts("The names are not the same"):
```

קומפיילרים מסוימים מחזירים מספר שלילי אם המחרוזת הראשונה "קטנה", מבחינה אלפביתית, מהמחרוזת השנייה, וערך חיובי אם המחרוזת הראשונה "גדולה" מהשנייה.

תרשים 10.8
המבנה של הפונקציה ()strcmp.



אם הקומפיילר שברשותך אינו כולל פקודת ()strcmp, באפשרותך לכתוב פונקציה משלך שמשווה בין שתי מחרוזות, פריט אחר פריט, בדומה למערכים מקבילים, ומפסיקה את התהליך כאשר מתגלה צמד שאינו זהה:

```
main()
{
int index, flag;
char name[10], name1[10];
gets(name);
gets(name1);
flag=0;
```

```
for(index=0;index<10;index++)
if(name[index]!=name1[index])
{
    flag=1;
    break;
}
if(flag==1)
puts("The strings are not the same");
else
puts("The strings are the same");
}
```

קביעת אורך המחרוזת

אורכה של המחרוזת אינו זהה בהכרח לגודלו של המערך. לדוגמה, באפשרותך להכריז על מערך בן 20 פריטים ששמו name, אך להזין שם הכולל פחות מ-20 תווים. מרבית הקומפיילרים של שפת C ושפת C++ עושים שימוש בפונקציה strlen() כדי לקבוע את מספר התווים בפועל במחרוזת, כפי שמודגם להלן:

```
gets(name);
count=strlen(name);
printf("The string %s has %d characters", name, count);
```

הפונקציה מקצה את מספר התווים שבקלט המחרוזת (המחרוזת main - בדוגמה שלנו) למשתנה מטיפוס מספר שלם (count). כדי לבצע אותה פעולה באופן ידני, השתמש ברוטינה כדוגמת:

```
main()
{
int index;
char name[10];
gets(name);
for(index=0;index<10;index++)
{
if(name[index]=='\0') break;
}
printf("the length is %d", index);
}
```

הרוטינה הזו סורקת את מערך התווים ומחפשת את מסיים האפס. מיקומו של מסיים האפס מציין את מספר התווים במחרוזת.

כדוגמה לשימוש בפונקציה לקביעת אורך מחרוזת, להלן תוכנית המציגה את היפוכה של מחרוזת:

```
main()
{
char name[10];
int index, count;
gets(name);
count=strlen(name);
for(index=count;index>0;index--)
    putchar(name[index-1]);
putchar('\n');
}
```

התוכנית עושה שימוש באורכה של המחרוזת בתור ערך התחלתי של אינדקס הכלול בלולאת for וקטן בכל חזרה על הלולאה. אם המחרוזת היא בת 5 תווים, הלולאה חוזרת 5 פעמים, והאינדקס מונה מ-5 ל-1. מכיוון שפריטי המערך ממוספרים מ-4 עד 0, מופחת 1 ממספר האינדקס כדי ליצור את המספר הסידורי.

הקצאת ערכים למחרוזות

שפת C אינה מאפשרת להקצות תווים למחרוזת באופן ישיר, כמו בביטוי:

```
name="Sam";
```

במקום זאת, ניתן להשתמש בפונקציה ()strcpy הכלולה במרבית הקומפיילרים. המבנה של הפונקציה הוא כדלקמן:

```
strcpy(name,"Sam");
strcpy(name, name1);
```

בדוגמה הראשונה, התווים Sam מוצבים במחרוזת ששמה name. בדוגמה השנייה, התווים שכבר הוקצו למשתנה ששמו name1 - מוצבים במשתנה ששמו name.

כדי לבצע את הפונקציה הזו בעצמך, עליך להקצות תווים לפריטי המערך הנפרדים:

```
char name[]="Alan";
main()
{
char person[10];
```

```
int count, index;
count=strlen(name);
for(index=0;index<=count;index++)
    person[index]=name[index];
puts(person);
}
```

הרוטינה מקצה תווים המוצבים במערך אחד לפריטים מקבילים במערך אחר.

איחוד מחרוזות

כאשר מאחדים שתי מחרוזות, מתווספים התווים של מחרוזת אחת לקצה המחרוזת האחרת, ומסיים האפס עובר לקצה המחרוזת החדשה שנוצרה. התהליך נקרא concatenation (שרשור). הפונקציה ()strcat (ראה תרשים 10.9) מוגדרת בתקן K&R. התווים הכלולים בפרמטר השני מתווספים לקצהו של הפרמטר הראשון.

תרשים 10.9
המבנה של הפונקציה ()strcat.

```
strcpy(name,"Adam");
strcpy(name1,"and Eve");
strcat(name, name1);
puts(name);
```

איחוד המשתנים name ו-name1:
"Adam" + "and Eve"

Adam and Eve

הערות ++C

שפת ++C מאפשרת העמסת יתר של אופרטורים, כלומר - ניתן לאחד שתי מחרוזות בעזרת האופרטור +, בהתאם לפורמט הבא:

```
new_string=string1+string2;
```

מערכי מחרוזות

ניתן ליצור מערכים של מחרוזות, בדיוק כפי שניתן ליצור מערכים של כל טיפוס נתונים אחר. אולם מערך של מחרוזות הוא למעשה מערך של מערכי תווים. מערך שפריטיו הם מערכים נקרא מערך דו-מימדי.

ניתן לדמות מערך דו-מימדי לגיליון נתונים בעל שורות ועמודות. יש להגדיר מערך כזה בעזרת שתי סדרות של מספרים סידוריים - סדרה אחת מציינת את מספרי השורות בגיליון, והשנייה את מספרי הטורים. ההוראה הבאה מכריזה על מערך בן 10 שורות ו-20 עמודות, הכולל 200 משתנים מטיפוס מספר שלם:

```
int table[10][20]
```

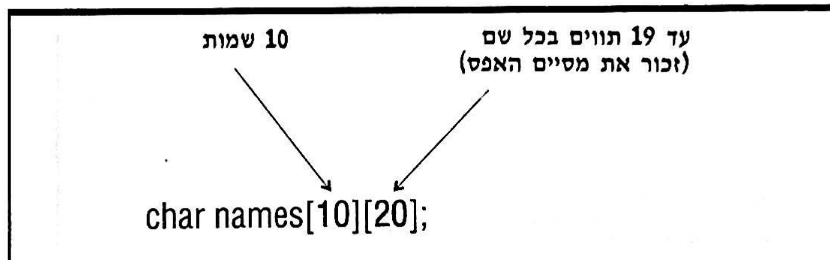
ניתן לדמיין כל פריט במערך כמספר שלם הנתון בתא משלו, בתוך גיליון נתונים של 20 X 10. הפריט table[0][0] ממוקם בתא השמאלי-עליון, ואילו הפריט table[0][1] ממוקם בתא שמימינו.

גם בהכרזה על מערך של מחרוזות יש להשתמש בשני מספרים סידוריים. הראשון מייצג את המספר המירבי של מחרוזות במערך, ואילו השני מייצג את האורך המירבי של כל מחרוזת. כלומר, אם מכריזים:

```
char names[10][20];
```

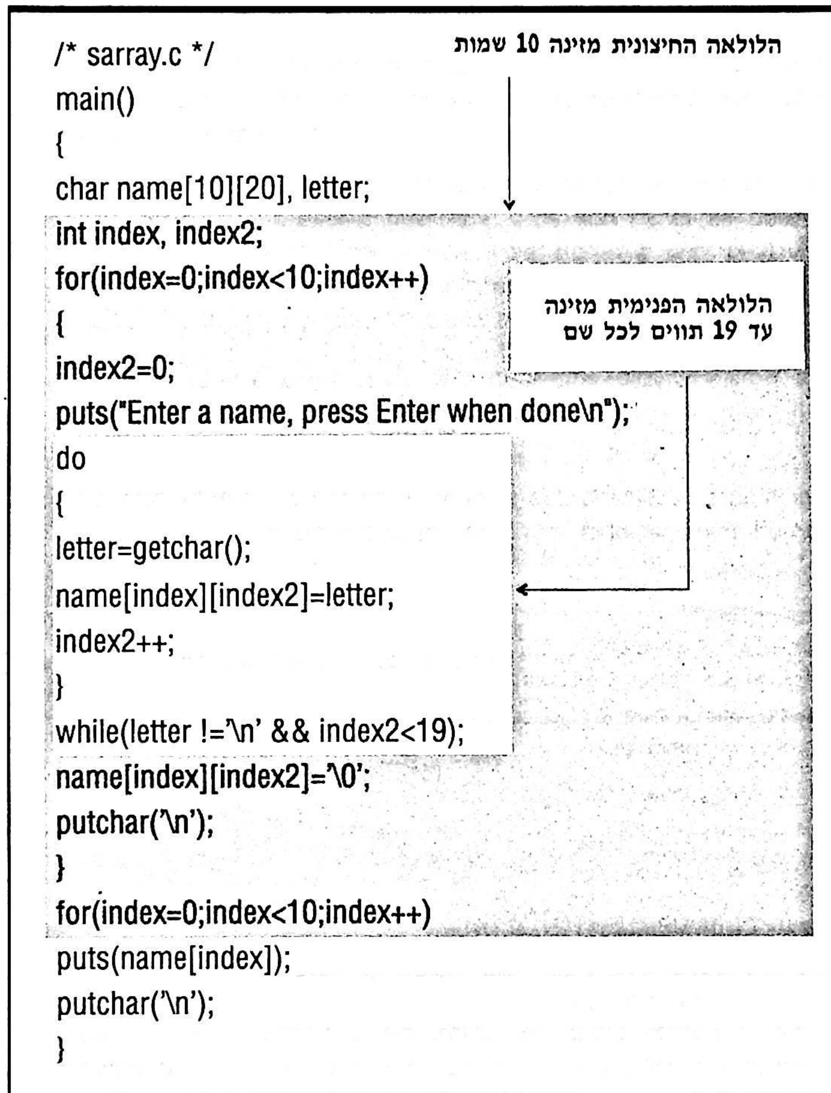
ניתן להזין לתוכו עשרה שמות, כשכל שם יכול להכיל עד 19 תווים (ראה תרשים 10.10).

תרשים 10.10
הכרזה על מערך מחרוזות.



כאשר עלינו להזין מחרוזות כתווים נפרדים, אנו זקוקים ללולאות מקננות. הלולאה החיצונית תחזור 10 פעמים, פעם אחת עבור כל מחרוזת. הלולאה הפנימית תחזור עד 19 פעמים, כדי להזין את התווים של כל מחרוזת, כפי שמודגם בתרשים 10.11. התוכנית מזינה 10 שמות לתוך מערך, ועושה שימוש בלולאת for כדי להציג את כל עשרת השמות.

תרשים 10.11
תכנות מערך מחרוזות.




```
/* sarray.c */
main()
{
char name[10][20], letter;
int index, index2;
for(index=0;index<10;index++)
{
index2=0;
puts("Enter a name, press Enter when done\n");
do
{
letter=getchar();
name[index][index2]=letter;
index2++;
}
while(letter !='\n' && index2<19);
name[index][index2]='\0';
putchar('\n');
}
for(index=0;index<10;index++)
puts(name[index]);
putchar('\n');
}
```

כדי להקצות תו למחרוזת, יש להשתמש בשני המספרים הסידוריים:

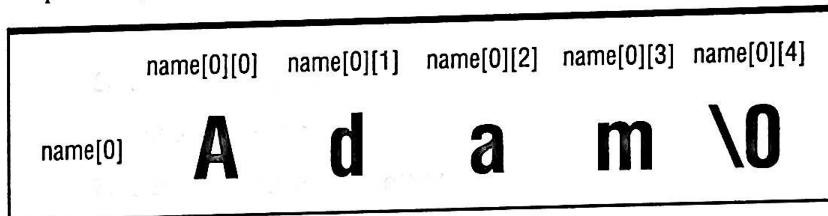
```
name[index][index2]=letter;
```

המספר הסידורי הראשון מציין את המחרוזת הרצויה במערך; המספר הסידורי השני מייצג את מיקומו של התו במחרוזת. לדוגמה, בתרשים 10.12, פריט name[0][3] הוא התו הרביעי

(m) במחרוזת הראשונה (Adam) במערך השמות. כדי להציג את המחרוזת כולה, יש צורך במספר סידורי אחד בלבד, שמייצג את המחרוזת עצמה:

```
puts(name[index]);
```

תרשים 10.12
מחרוזות ופריטים במערך מחרוזות.



למרבה המזל, ניתן להזין מחרוזות שלמות כיחידות, בעזרת לולאה אחת בלבד שחוזרת פעם אחת עבור כל מחרוזת במערך:

```
main()
{
char name[10][20];
int index;
for(index=0;index<10;index++)
gets(name[index]);
for(index=0;index<10;index++)
puts(name[index]);
}
```

עיצוב התוכנית

מכיוון שיש להכריז על גודלו המירבי של המערך בתחילת התוכנית או בתחילת הפונקציה, זוהי אחת ההחלטות הראשונות שיש לקבל. אם גודלו של המערך קטן מהגודל הדרוש בעת הרצת התוכנית, תתקבל שגיאת הרצה או תוצאות בלתי עקביות.

מאידך, יש להימנע מהכרזה על מערך תוך שימוש במספרים עצומים, "כדי להיות בטוח". מערכים, ובמיוחד מערכים של נתונים מטיפוס float ומערכי מחרוזות, הם זוללי זיכרון אמיתיים. ככל שהתוכניות שלך נעשות גדולות ומורכבות יותר, הכרזה על מערכים עצומים שאין בהם צורך, עלולה להביא לשגיאות של מחסור בזיכרון.

תכנן בקפידה את גודלו של המערך. השאר לעצמך מרווח שדי בו כדי לוודא שהמספר הסידורי אינו עולה על מספר הפריטים המירבי.

שאלות

1. מהו מערך?
2. האם מערך יכול לאחסן יותר מטיפוס אחד של משתנים?
3. כיצד מכריזים על מערך?
4. מהו מערך דו-מימדי?
5. כיצד מכריזים על מערך דו-מימדי?
6. מהו הקשר בין ערכו של המספר הסידורי לבין ערכו של פריט המערך?
7. כיצד משווים בין שתי מחרוזות?
8. כיצד מקצים ערך למשתנה מחרוזת?

תרגילים

1. כתוב תוכנית שעושה שימוש במערכים כדי לאחסן את השמות, הכתובות ומספרי הטלפון של 20 אנשים. אחסן את השמות לפי הסדר: שם פרטי, שם משפחה.
2. שנה את התוכנית שכתבת בתרגיל 1, כך שניתן יהיה להזין שם ולאתר במערכים את מספר הטלפון שלו.
3. הסבר מה לקוי בתוכנית הבאה:

```
main()
{
int temps(31);
int index, total;
for(index=0; index < 31; index++)
    {
    printf("Enter temperature #%d:", index);
    scanf("%d",&temps(index));
    }
    high=temps(0);
    low=temps(0;
```



```
index=1;
while(index< 31)
    {
    if(temps(index)>high)
        high=temps(index);
    else
    low=temps(index);
    index++;
    }
printf("low is %d\n",low);
printf("high is %d\n",high);
}
```

11

מבנים ומצביעים

כל המשתנים שהכרנו עד עתה הכילו ערכים מטיפוס נתונים אחד בלבד: תווים, מחרוזות, מספרים שלמים או מספרים עשרוניים. אפילו מערך, בין אם הוא חד-מימדי או דו-מימדי, יכול לאחסן רק משתנים מטיפוס אחד. אלו הם משתנים פשוטים, כלומר - ניתן להקצות להם ערך בצורה פשוטה, על ידי איזכור שמם בהוראת קלט או בהוראת הקצאה.

בפרק זה תכיר שני סוגים של משתנים רב-צדדיים יותר. משתנה מבנה הוא אסופה של משתנים אחרים, המכילים ערכים מטיפוסים שונים. מצביעים הם משתנים המתייחסים למיקומם בזיכרון של משתנים אחרים.

מבנים ומצביעים מאפשרים פיתוח של יישומים מורכבים יותר.

שימוש במבנים

תוכנית מחשב מייצגת בדרך כלל מצב מציאותי כלשהו, שמצריך ארגון ועיבוד של מידע באופן שיטתי: מערכת הנהלת חשבונות, בעיה הנדסית, מערכת רישום תלמידים או סדרת פעולות אחרת שניתן לבצעה באמצעות אלגוריתמים של תוכניות מחשב.

בתוך התוכנית אנו מייצגים כל פיסת מידע במשתנה, ומקצים אותו לטיפוס נתונים מסוים: float, char, int או מחרוזת. אך במציאות עלינו לטפל, לעתים קרובות, בעצמים שמכילים יותר מטיפוס אחד של נתונים. לדוגמה, נניח שאתה מקטלג את אוסף תקליטי הקומפקט-דיסק שלך. ברשותך קבוצה של כרטיסים, כשכל כרטיס מכיל את שמו של תקליט קומפקט-דיסק באוסף, תיאורו, מחירו, סוג המוסיקה המוקלטת עליו ומספרו הסידורי (תרשים 11.1).

תרשים 11.1
תיעוד בלתי-ממוחשב.

הערות ++C

בשפת ++C, מבנה יכול לשמש גם להגדרת מחלקה (class). מחלקה היא מבנה שמכיל משתנים ופונקציות. על ידי הגדרת הפונקציה בתוך המחלקה אנו משייכים אותה למשתני המבנה, כמו בקטע התוכנית הבא:

```
struct square {
        float number;
        void assignnumber(double);
        float squareit(void);
} amount;
void square::assignnumber(float num)
{
        number = num;
}
void square::squareit(void)
{
        float toreturn;
        toreturn = number*number;
        return (toreturn);
}
square.assignnumber(25.0);
cout << "The square of the number is " <<
square.squareit();
```

הסימן :: בהגדרות הפונקציות נקרא אופרטור scoping, ומשמש לשיוך הפונקציה למחלקה.

ניתן למחשב את הקטלוג בקלות על ידי הקצאת כל אחד מהפריטים למשתנה:

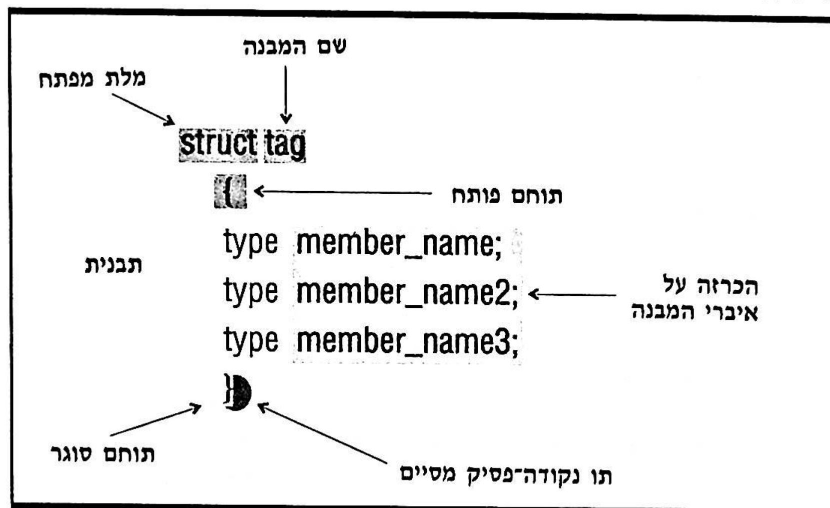
```
char name[20], description[40], category[12];
float cost;
int number;
```

אולם במציאות אנחנו לא מטפלים בחמשת פריטי המידע האלה בנפרד, אלא מאחסנים אותם כקבוצה על-גבי הכרטיס. אומנם כל פיסת מידע על-גבי הכרטיס שונה מחברתה, אך יחד הן מייצגות עצם אחד - תקליט קומפקט-דיסק אחד באוסף.

אם ברצוננו לכתוב תוכנית שתייצג בצורה נאמנה את המצב המציאותי, אנו זקוקים לדרך כלשהי שתאפשר לנו לטפל בטיפוסים שונים של נתונים כאסופה. למרבה המזל, שפת C ושפת ++C מאפשרות לאגד טיפוסי נתונים שונים לעצם אחד שנקרא *מבנה*, הגרסה הממוחשבת לכרטיס קטלוגי.

הגדרת מבנה

השלב הראשון ביצירת אסופה של משתנים הקשורים זה לזה הוא הגדרת המבנה. הגדרת מבנה היא למעשה הגדרה של טיפוס נתונים משלך. אתה מעניק למבנה שם, ומוסר לקומפיילר את השם ואת טיפוס הנתונים אליו משתייך כל נתון שברצונך לאחסן במבנה, כפי שמודגם בתרשים 11.2.



תרשים 11.2
מרכיבי המבנה.

על ההגדרה להתחיל במלת המפתח struct, ולאחריה שמו של המבנה, המכונה גם tag (תווית). הנתונים היוצרים את המבנה נקראים *איברי המבנה*, ומוצבים בקבוצה בין סוגריים מסולסלים.

ההכרזה על איברי המבנה דומה להכרזה על משתנים. יש לציין את טיפוס הנתונים של כל איבר, וכן את גודלם של מערכים או מחרוזות כלשהם. כמו כן, יש להציב נקודה-פסיק אחרי כל הכרזה, וכן בסופו של המבנה.

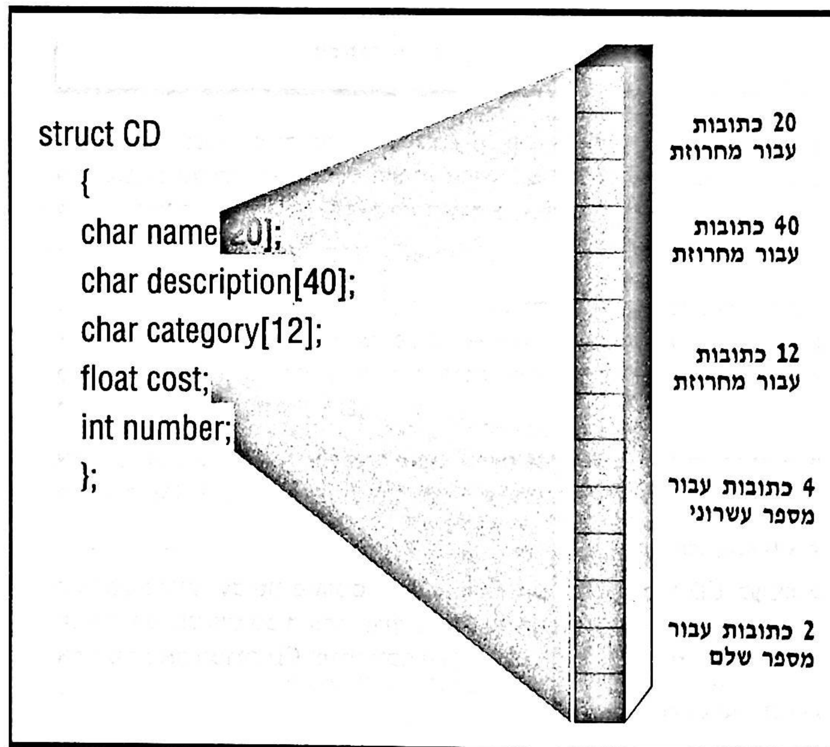
רשימת איברי המבנה נקראת template (תבנית). המבנה אינו מכריז למעשה על משתנים. איברי המבנה עצמם אינם משתנים, אלא מרכיבים של משתנה, אחד או יותר (הנקראים משתני מבנה), שיוכרז כמשתייך לטיפוס המבנה. התבנית מגדירה את המרכיבים ומורה לקומפיילר כמה זיכרון יש להקצות עבור כל משתנה מבנה.

ניתן להגדיר את אוסף תקליטי הקומפקט-דיסק בעזרת המבנה הבא:

```
struct CD
    {
    char name[20];
    char description[40];
    char category[12];
    float cost;
    int number;
    };
```

סדרת הוראות זו יוצרת טיפוס נתונים חדש ששמו CD, המכיל חמש פיסות מידע מטיפוסים שונים: שלוש מחרוזות, מספר עשרוני ומספר שלם. המבנה מקצה אזור בזיכרון שרוחבו 78 עמדות זיכרון (תרשים 11.3).

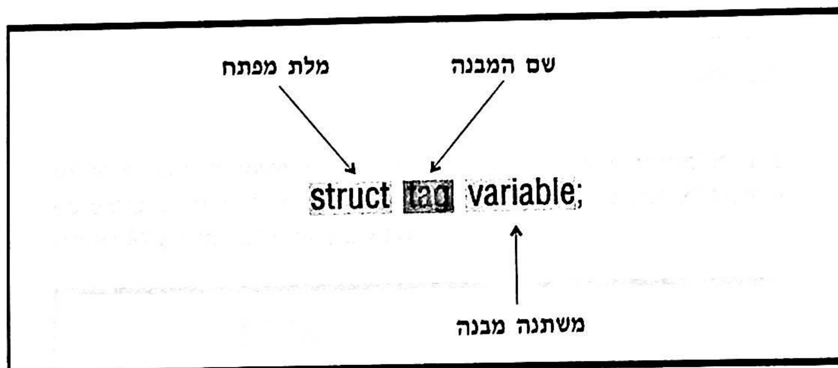
תרשים 11.3
מבנה מקצה שטח בזיכרון עבור איברי המבנה.



הקצאת ערכים למשתנה מבנה

הגדרת המבנה עדיין לא מעניקה לנו גישה לעמדות הזיכרון. כזכור, קודם שניתן יהיה להשתמש בטיפוס נתונים כלשהו, יש להכריז על משתנה המשתייך לאותו טיפוס נתונים. לא ניתן להשתמש בערך מטיפוס float, למשל, לפני שמכריזים על משתנה מטיפוס float. באופן דומה, לא ניתן להשתמש במבנה עד שמכריזים על משתנה המשתייך לטיפוס הנתונים של המבנה, באופן המתואר בתרשים 11.4.

תרשים 11.4
הכרזה על משתנה מבנה.



מלת המפתח struct מורה לקומפיילר שהכוונה היא לטיפוס מבנה, ותווית המבנה מזהה את התבנית של המשתנה. אחרי התווית מופיע שמו של המשתנה בו תשתמש בתוכנית. לדוגמה, ניתן לגשת לאסופה CD על ידי הגדרת משתנה כך:

```
struct CD disc;
```

כעת יש ברשותנו משתנה ששמו disc, המהווה אסופה של חמישה פריטים. כמו כל משתנה אחר, disc מייצג אזור בזיכרון. אך מכיוון ש-disc הוא משתנה מבנה מטיפוס CD, הוא מייצג אזור בזיכרון שרוחבו 78 כתובות, שיכיל שלוש מחרוזות, ערך מטיפוס float וערך מטיפוס מספר שלם (תרשים 11.5).

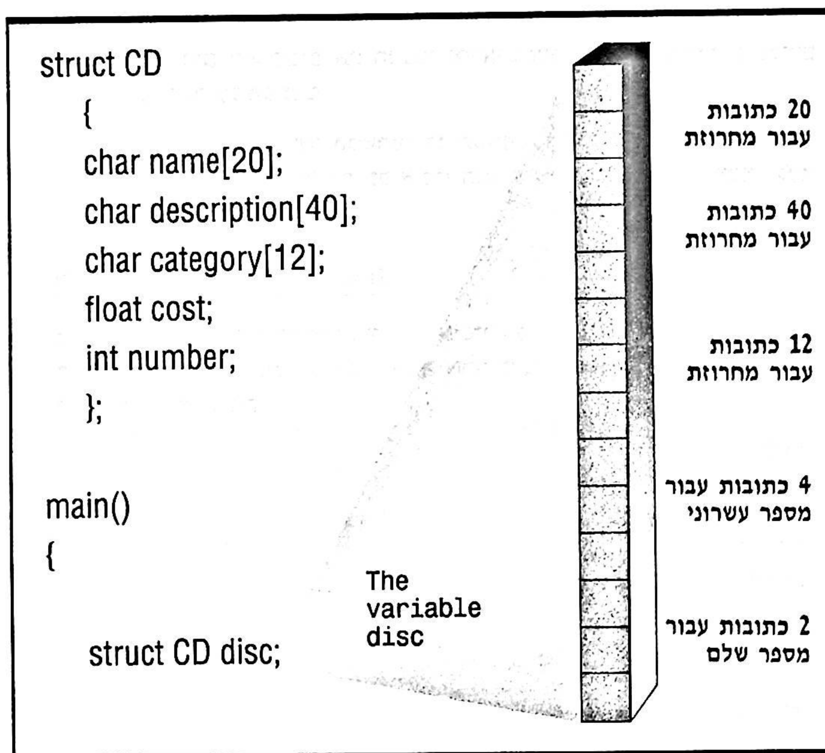
אם אתה זקוק ליותר ממשתנה אחד מאותו טיפוס (לדוגמה, כדי לתעד נתונים דומים עבור תקליטורי CD-ROM שברשותך), ניתן להכריז עליהם בהוראה אחת:

```
struct CD disc, cdrom;
```

ההוראה מכריזה על שני משתנים (disc ו-cdrom) מטיפוס המבנה CD. לעומת זאת, אם הגדרת יותר מטיפוס מבנה אחד, עליך להכריז על המשתנים של כל טיפוס בנפרד. לדוגמה, אם הגדרת את המבנה CD ומבנה נוסף ששמו VIDEO, עליך להכריז על המשתנים כך:

```
struct CD disc, cdrom;
struct VIDEO movies, vacation;
```

תרשים 11.5 מרכיביה של הכרזת משתנה מבנה.



ניתן גם ליצור את המבנה ולהכריז על המשתנים בצעד אחד. הצב את שם המשתנה (או שמות המשתנים) בין התוחם הסוגר את המבנה לבין תו הנקודה-פסיק, כך:

```
struct CD
    {
    char name[20];
    char description[40];
    char category[12];
    float cost;
    int number;
    } disc;
```

כדי להבהיר את המינוח, הבה נסקור שוב את מרכיבי ההכרזה:

- CD היא *תווית המבנה*. זהו שמו של טיפוס נתונים חדש, אסופה של פריטים הנקראת מבנה.

- הפריטים המרכיבים את המבנה נקראים *איברי המבנה*. ההכרזה עליהם דומה להכרזה על משתנים.

- Disc הוא שמו של המשתנה בו תשתמש בתוכנית. זהו משתנה מטיפוס CD, כלומר - הוא מכיל את כל איברי המבנה שהוכרזו כחלק מהמבנה. לעתים הוא מכונה גם *משתנה מבנה*.

הקצאת ערכים התחלתיים

כאשר ידועים הערכים ההתחלתיים של איברי המבנה, ניתן להקצות את הערכים בעת הכרזת המשתנה. אם אנו יוצרים משתנה אחד בלבד מטיפוס המבנה, ניתן לאתחל אותו כחלק מהגדרת המבנה:

```
struct CD
    {
    char name[20];
    char description[40];
    char category[12];
    float cost;
    int number;
    } disc = {"Best Hits","Tiny Tim","pop music", 12.50, 12};
```

סדרת הוראות זו יוצרת את המבנה CD, מכריזה על המשתנה disc ומקצה ערך התחלתי לכל אחד מחמשת איברי המבנה (תרשים 11.6).

אפשרות אחרת היא לאתחל את איברי המבנה בהכרזת המשתנה:

```
struct CD disc = {"My Life and Times","B. Gates biography",
            "book on disc", 24.99, 213};
```

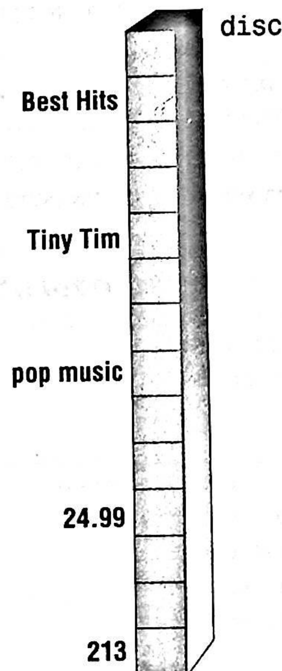
מכיוון ששפת C היא בעלת פורמט חופשי, ניתן לחלק את פקודת ההקצאה למספר שורות, כל עוד לא מחלקים מחרוזת הנתונה בתוך מרכאות.

ניתן להגדיר מבנה חיצוני לפני הפונקציה ()main, או להגדיר מבנה מקומי בתוך ()main או בתוך פונקציה אחרת. עם זאת, כאשר מקצים ערכים התחלתיים למבנה שמכיל מחרוזות, יש להכריז עליו לפני הפונקציה ()main, או כמשתנה סטטי:

```
static struct CD
```

תרשים 11.6
אתחול משתנה מבנה.

```
struct CD
    {
    char name[20];
    char description[40];
    char category[12];
    float cost;
    int number;
    } disc = {"Best Hits",
              "Tiny Tim",
              "pop music",
              24.99, 213}
```



שימוש במבנה

איבר הוא תמיד חלק ממבנה. לא ניתן להתייחס אל איבר כשלעצמו. לדוגמה, אם ננסה להקצות ערך לאיבר cost במבנה CD באמצעות הביטוי:

```
cost = 23.13;
```

הקומפיילר יציג הודעת שגיאה, מכיוון שהמשתנה cost אינו מוגדר. לא הוקצה זיכרון תחת שם המשתנה הזה. המשתנה בו אנו מעוניינים הוא disc, שכולל איבר ששמו cost. כדי להתייחס לאיבר, יש לציין אותו באמצעות שמו של משתנה המבנה, כך:

```
structure-variable.member-name
```

כלומר, יש לציין את שמו של משתנה המבנה, נקודה, ולאחריה את שמו של האיבר, כמו בהוראות הבאות:

```
gets(disc.name);
gets(disc.description);
gets(disc.category);
disc.cost = 16.95
disc.number = 5
```

צורת הכתיב הזו מורה לקומפיילר היכן בדיוק למקם את הערך בזיכרון; למשל, באיבר name של משתנה המבנה disc.

גם פלט של ערכי איברים יש ליצור בצורה דומה, עם נקודה בין משתנה המבנה לבין שם האיבר. תוכנית 11.1, לדוגמה, מזינה ומציגה מידע אודות המבנה CD. התייחסות לשמות האיברים מלווה תמיד במשתנה המבנה.

ניתן להזין וליצור פלט של פריטי המבנה בכל סדר שהוא; אין הכרח להשתמש בהם בסדר בו הם מוגדרים בתבנית המבנה.

תוכנית 11.1: שימוש במבנה עם קלט ופלט

```
/*CD1.C*/
struct CD
{
char  name[20];
char  description[40];
char  category[12];
float cost;
int  number;
} disc;

main()
{
     puts("Enter disk information\n\n");
     printf("Enter the name:");
     gets(disc.name);
     printf("Enter the description:");
     gets(disc.description);
     printf("Enter the category:");
     gets(disc.category);
     printf("Enter the cost:");
     scanf("%f", &disc.cost);
     printf("Enter the slot number:");
     scanf("%d", &disc.number);
     puts("The information on the CD is:\n\n");
     printf("Name:          %s\n",disc.name);
     printf("Description:   %s\n",disc.description);
     printf("Category:      %s\n",disc.category);
     printf("Cost:          %6.2f\n",disc.cost);
     printf("Location:      %d\n",disc.number);
}
```

מערכי מבנה

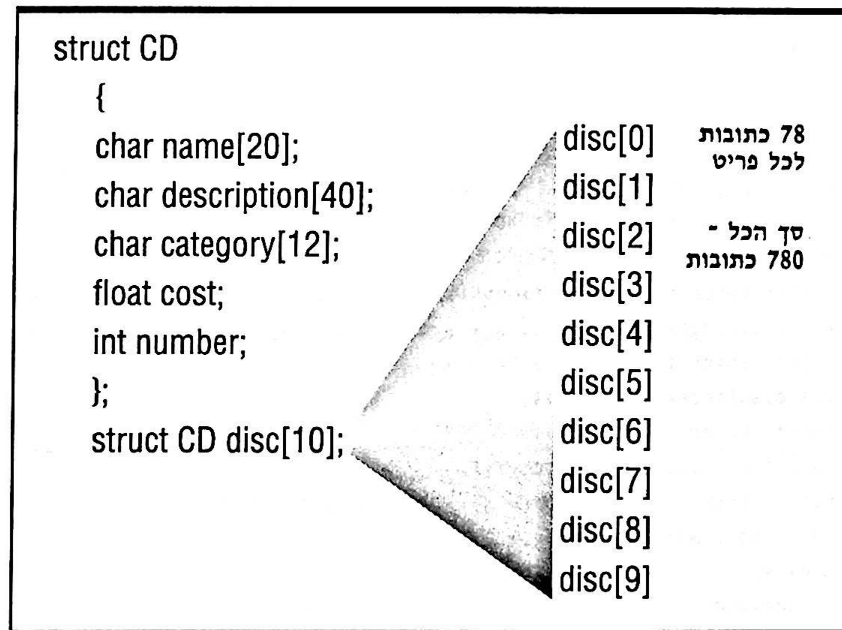
עד עתה עבדנו עם המקבילה של כרטיס אחד בלבד; אולם הגדרנו את המבנה כדי שנוכל לטפל בנתונים של אוסף תקליטי הקומפקט-דיסק כולו. עלינו להשתמש, אם כן, במערך - במקום במשתנה יחיד. כאשר אנו יוצרים מערך של מבנה, אנו יוצרים מערך של התבנית כולה. יש להכריז על המערך בעזרת מספר סידורי המוצמד לשמו של משתנה המבנה:

```
struct CD disc[10];
```

הוראה זו מקצה עשרה מקטעי זיכרון, כשגודלו של כל אחד מהם מספיק לאחסון מבנה שלם (תרשים 11.7). התייחסות לפריט מערך מתבצעת על ידי הצמדת המספר הסידורי למשתנה המבנה, ולא לשם האיבר:

```
gets(disc[0].name);
gets(disc[1].name);
```

תרשים 11.7
מערך של מבנה.



תוכנית 11.2, למשל, עושה שימוש במערך מבנה כדי לייצג את המטלה המציאותית של מילוי סדרה של כרטיסים עבור אוסף תקליטי הקומפקט-דיסק.

תוכנית 11.2: תכנות מערך מבנה

```
/*CD2.C*/
struct CD
    {
    char  name[20];
    char  description[40];
    char  category[12];
    float cost;
    int  number;
    } disc[10];

main()
{
int index, repeat;
char flag;
flag =  'Y';
index = 0;
do
  {
    printf("Enter disk # %d\n",index);
    printf("Enter the name:");
    gets(disc[index].name);
    printf("Enter the description:");
    gets(disc[index].description);
    printf("Enter the category:");
    gets(disc[index].category);
    printf("Enter the cost:");
    scanf("%f", &disc[index].cost);
    printf("Enter the slot number:");
    scanf("%d", &disc[index].number);
    index++;
    if(index<10)
     {
    printf("Do you have another? Y or N");
scanf("%C", &flag);
    }
```

```
 }
while (index <10 && (flag == 'Y' || flag == 'y'));
puts("Name            Slot Number");
for(repeat=0; repeat<index;repeat++)
printf("%s    %d\n",disc[repeat].name,  disc[repeat].number);
}
```

מערך בן 10 פריטים, הנקרא disc, מוכרז כחלק מהגדרת המבנה:

```
} disc[10];
```

המשתנה flag מוכרז כדי לציין את סיום הקלדת המידע על ידי המשתמש, במקרה שברשותו פחות מ-10 תקליטים. לולאת ה-do מזינה את פריטי המערך: חמישה נתונים (איברי מבנה) לכל אחד מפריטי המערך (תקליטו קומפקט-דיסק באוסף). המשתנה index ישמש מספר סידורי.

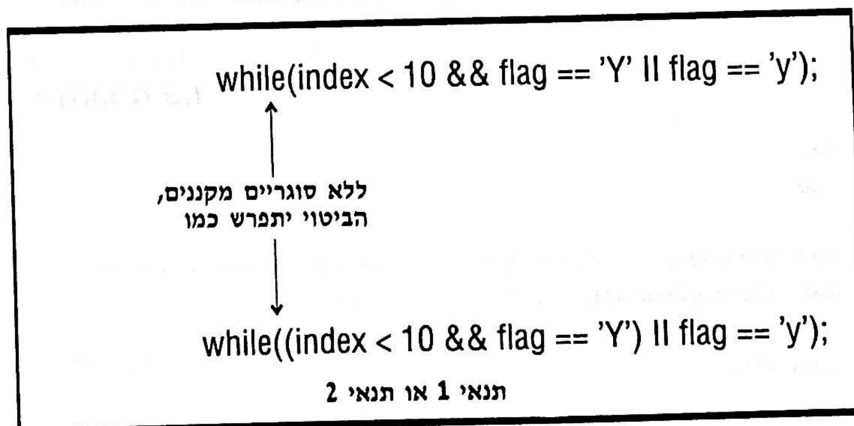
אחרי הזנת כל פריט מערך, מוצגת למשתמש השאלה אם ברשותו תקליט קומפקט-דיסק נוסף. עם זאת, השאלה אינה מוצגת אם המשתמש הזין כבר את כל 10 פריטי המערך.

לולאת ה-do משתמשת בתנאי הבא:

```
while (index <10 && (flag == 'Y' || flag == 'y'));
```

השימוש באופרטור **וגם** פירושו ששני התנאים חייבים להתקיים: index קטן מ-10, וגם המשתנה flag שווה Y או y.

ללא הסוגריים המקננים, קומפיילר C היה מאחד את שני התנאים הראשונים ומתייחס להוראה כתנאי **או** (כפי שמודגם בתרשים 11.8). הלולאה היתה ממשיכה לבצע חזרות גם אחרי שהוזנו כל עשרת הפריטים, אם ערכו של המשתנה flag היה y מהקלט האחרון.



תרשים 11.8 אופן התרגום ללא סוגריים מקננים.

כאשר מזינים את נתוני התקליט האחרון, ערכו של index גדול ב-1 ממספר פריטי המערך. המשתנה משמש בלולאת for כדי להדפיס רשימה של התקליטים ומיקומם.

שים לב שכל הפניה לאיבר מבנה כוללת את שם המשתנה. המספר הסידורי של המערך מופיע אחרי שם המשתנה.

מבנים ופונקציות

הגדרת K&R C המקורית הגבילה את השימוש במבנים. ניתן היה להעביר מבנים כארגומנטים רק בעזרת מצביעים מיוחדים, עליהם תלמד בהמשך הפרק. בנוסף, לא ניתן היה להקצות באופן ישיר מבנה אחד למבנה אחר, כך:

```
CDROM = disc;
```

הקומפיילרים המודרניים של C ושל ++C התגברו על המגבלה הזו. ניתן להקצות ישירות מבנה אחד למבנה אחר, ולהעביר מבנה שלם לפונקציה, או לקלוט מבנה שלם מפונקציה.

במרבית הקומפיילרים של ++C ו-ANSI C, למשל, אפשר להעביר ולהחזיר מבנה שלם. תוכנית 11.3, לדוגמה, מדגימה כיצד להעביר מבנה לפונקציה. ערכי האיברים מוזנים בפונקציה ()main, ולאחר מכן מועבר המבנה כולו אל הפונקציה ()putdisc לצורך פלט. משתנה המבנה משמש בקריאה לפונקציה:

```
putdisc(disc);
```

הוראה זו מעבירה את המבנה כולו אל הפונקציה. הפונקציה מכילה משתנה שיקבל את המבנה המועבר, ומכריזה על המשתנה כמשתייך לאותו טיפוס מבנה:

```
putdisc(disk);

struct CD disk;
```

הפונקציה מכירה עתה משתנה מבנה נפרד, disk, המכיל את האיברים המועברים של המבנה disc. בתוך הפונקציה, הפניה לאיברים היא באמצעות התווית disk, שמו של המבנה הקולט.

תוכנית 11.3: העברת מבנה



הערה
קומפיילר PCC המצורף לספר זה מציג אזהרה כאשר תוכנית מעבירה מבנה מפונקציה אחת לאחרת. ניתן להתעלם מאזהרה זו, והיא תושמט בגרסאות הבאות של הקומפיילר.

```
/*CD3.C*/
struct CD
     {
     char  name[20];
     char  description[40];
     char  category[12];
     float cost;
     int  number;
```

```
    } disc;
main()

{
puts("Enter disk information\n\n");
printf("Enter the name:");
gets(disc.name);
printf("Enter the description:");
gets(disc.description);
printf("Enter the category:");
gets(disc.category);
printf("Enter the cost:");
scanf("%f", &disc.cost);
printf("Enter the slot number:");
scanf("%d", &disc.number);
putdisc(disc);
}


putdisc(disk)
struct CD disk;
{
puts("The information on the CD is:\n\n");
printf("Name:           %s\n",disk.name);
printf("Description:    %s\n",disk.description);
printf("Category:       %s\n",disk.category);
printf("Cost:           %6.2f\n",disk.cost);
printf("Location:       %d\n",disk.number);
}
```

תוכנית 11.4 מדגימה כיצד להחזיר מבנה מפונקציה. איברי המבנה מוזנים לפונקציה ()getdisc, ולאחר מכן מוחזרים לפונקציה ()main בפקודה:

```
return(inputdisc);
```

שים לב שתווית המבנה (CD) משמשת הן בהכרזת הפונקציה והן בהכרזת המשתנה של getdisc(). כאשר מחזירים טיפוס נתונים שאינו int או char, יש לציין את שם הטיפוס בהכרזת הפונקציה. טיפוס הנתונים המוחזר הוא טיפוס המבנה CD, ולכן הפונקציה מוכרזת כך:

```
struct CD getdisc()
```

גם המבנה המשמש בפונקציה הוא מטיפוס CD, ולכן הוא מוכרז כך:

```
struct CD inputdisc;
```

שים לב גם לכך שהפונקציה getdisc() מוכרזת באופן גלובלי כמשתנה מבנה, יחד עם המשתנה disc. יש צורך בהכרזה מסוג זה כדי שהפונקציה תחזיר מבנה.

תוכנית 11.4: החזרת מבנה

```
/*CD4.C*/
struct CD
     {
     char  name[20];
     char  description[40];
     char  category[12];
     float cost;
     int  number;
     } disc, getdisc();
main()
{
disc = getdisc();
puts("The information on the CD is:\n\n");
printf("Name:          %s\n",disc.name);
printf("Description:   %s\n",disc.description);
printf("Category:      %s\n",disc.category);
printf("Cost:          %6.2f\n",disc.cost);
printf("Location:      %d\n",disc.number);
}


struct CD getdisc()
{
     struct CD inputdisc;
     puts("Enter disk information\n\n");
```

```
    printf("Enter the name:");
    gets(inputdisc.name);
    printf("Enter the description:");
    gets(inputdisc.description);
    printf("Enter the category:");
    gets(inputdisc.category);
    printf("Enter the cost:");
    scanf("%f", &inputdisc.cost);
    printf("Enter the slot number:");
    scanf("%d", &inputdisc.number);
    return(inputdisc);
}
```

הבנת נושא המצביעים

בעוד שניתן להעביר לפונקציה כל מספר של ארגומנטים, ובכלל זה גם מבנה שלם, אפשר לכלול פרמטר אחד בלבד בפקודת ()return. כלומר, באפשרותך להחזיר ערך אחד בלבד אל הרוטינה הקוראת לפונקציה בתוכנית. פתרון אחד לבעיה הוא שימוש במשתנים גלובליים. אולם, בדרך זו מאבדים חלק מהבקרה על מבנה התוכנית, וקשה יותר לאתר שגיאות. לכן, אם ברצונך להחזיר יותר מערך אחד מפונקציה, השתמש במצביעים.

ישנן שתי דרכים לאזכר משתנה בתוכנית. כאשר משתמשים בשמו של המשתנה, ההפניה היא לערך המאוחסן בעמדת הזיכרון. כאשר משתמשים באופרטור הכתובת (&), ההפניה היא אל כתובת הזיכרון בה מאוחסן המשתנה.

כאשר מקצים ערך למשתנה, כמו בדוגמה הבאה:

```
tax = 35;
```

מחדירים ערך לעמדה בזיכרון. כתובת הזיכרון שהוקצתה למשתנה tax תכיל את הערך 35. ניתן גם להציג את *כתובת הזיכרון* של משתנה בעזרת האופרטור &. ההוראה:

```
printf("%d", &tax)
```

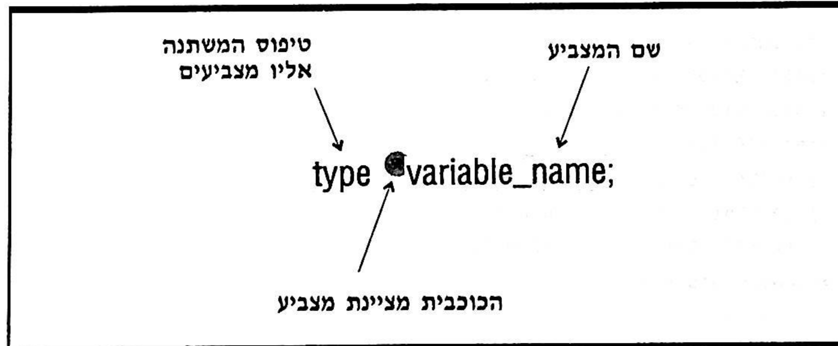
תציג את הכתובת בה מאוחסן המשתנה tax, ולא את הערך המאוחסן באותה כתובת. עם זאת, אופרטור הכתובת, כדוגמת &tax, יכול לשמש רק בביטוי. הוא אינו מהווה משתנה, מכיוון שהוא מייצג מספר שלם בלבד; הוראה כדוגמת

```
&tax = 25;
```

אינה הוראה תקפה.

מצביע הוא משתנה המכיל את כתובת הזיכרון של משתנה אחר. אם המשתנה tax מאוחסן בעמדה 21260, אזי המצביע למשתנה tax יכיל את המספר 21260.

תרשים 11.9
הכרזה על מצביע.



כדי ליצור משתנה מצביע, השתמש בצורת הכתיב המודגמת בתרשים 11.9. ראשית יש לציין את טיפוס הנתונים המאוחסן במיקום המזוהה על ידי המצביע. תו הכוכבית מורה לקומפיילר שמדובר במשתנה מצביע. לבסוף, יש לציין את שמו של המשתנה. לדוגמה, ההוראה

```
int *taxptr
```

יוצרת משתנה מטיפוס מצביע (מסומן באמצעות כוכבית) ששמו taxptr, שיכיל את כתובתו של משתנה מטיפוס int. ההוראה:

```
float *net
```

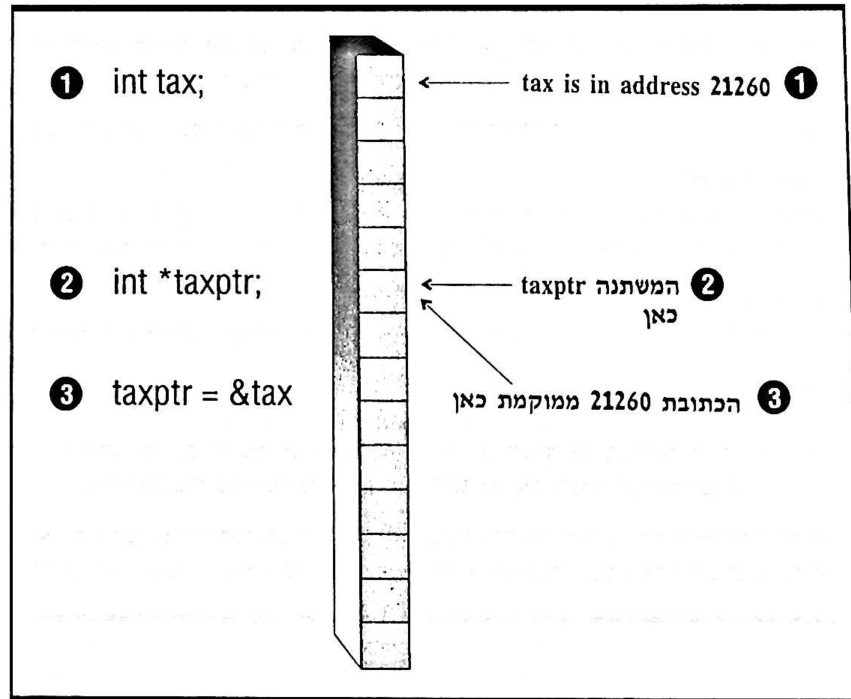
יוצרת מצביע ששמו net, שיכיל את כתובתו של משתנה מטיפוס float.

לאחר שמכריזים על משתנה מסוג מצביע, יש להצביע באמצעותו על דבר–מה. עושים זאת על ידי הקצאת כתובתו של המשתנה עליו יש להצביע, כך:

```
taxptr = &tax
```

הוראה זו מציבה במשתנה taxptr את הכתובת שבה מאוחסן המשתנה tax (כפי שמודגם בתרשים 11.10). אם tax מאוחסן בכתובת הזיכרון 21260, אזי ערכו של המשתנה taxptr הוא 21260.

תרשים 11.10
הקצאת כתובת למצביע.



לדוגמה, עיין בתוכנית הבאה:

```
main()
{
int *taxptr;
int tax;
taxptr=&tax;
tax=35;
printf("tax is %d\n", tax);
printf("the tax pointer is %d\n", taxptr);
}
```

אם נשתמש בנתוני המדגם שלנו, הפלט יהיה:

```
tax is 35
the tax pointer is 21260
```

בשלב זה ייתכן שאתה תוהה: "אז מה?" - אחרי הכל, ניתן להשיג תוצאה זהה על ידי הקצאת כתובתו של tax למשתנה רגיל מטיפוס מספר שלם (שאינו מצביע). היתרון הוא בכך שניתן

לאזכר את משתנה המצביע כ-*taxptr. הכוכבית מורה לקומפיילר שאנחנו מעוניינים *בתוכנה של כתובת הזיכרון המאוחסנת במצביע.* אין לנו עניין בערך 21260, אלא בערך שמאוחסן במיקום זה בזיכרון. הערך של taxptr הוא 21260, אולם הערך של *taxptr הוא 35.

למשתנה taxptr ניתן להקצות כתובת זיכרון בלבד. ההוראה

```
taxptr = 21260
```

תגרום לשגיאת קומפיילר מכיוון שהיא מנסה להקצות למצביע מספר שלם. ההקצאה היחידה שניתן לבצע למשתנה taxptr היא כתובתו של משתנה, בעזרת אופרטור הכתובת, כך:

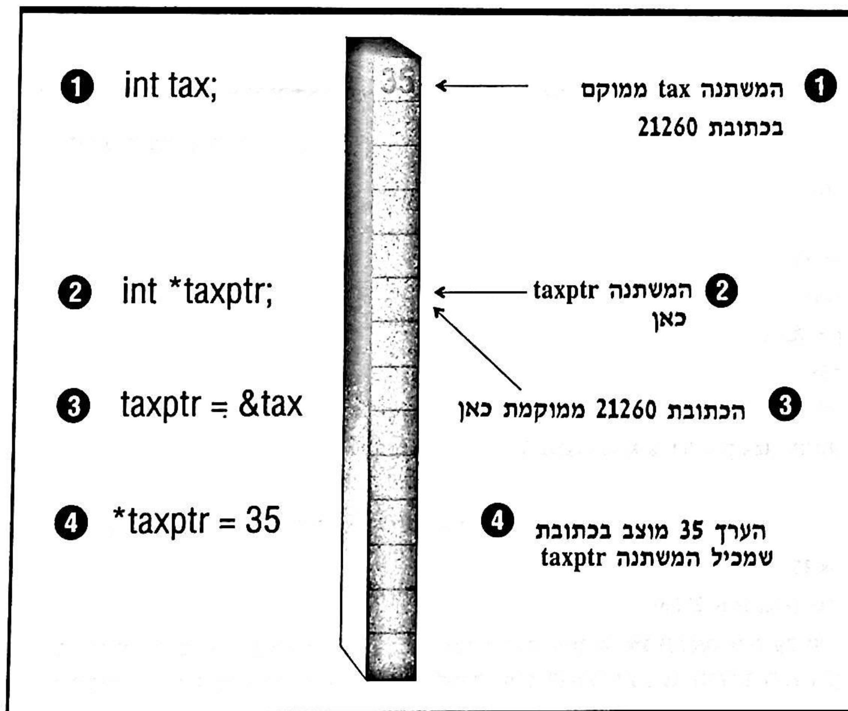
```
taxptr=&tax
```

עם זאת, ניתן להקצות ערך למצביע *taxptr, כמו בהוראה:

```
*taxptr = 35
```

הוראה זו פירושה "הצב את הערך 35 בכתובת הזיכרון עליה מצביע המשתנה taxptr". מכיוון שהמצביע מכיל את הכתובת 21260, מוצב הערך 35 בכתובת זיכרון זו. כמו כן, מכיוון שזוהי כתובתו של המשתנה tax, מקבל גם tax את הערך 35 (תרשים 11.11).

לכאורה, נראה שזוהי דרך מסורבלת למדי לשינוי ערך של משתנה. קל יותר להקצות ל-tax ערך חדש. אולם כוחם האמיתי של מצביעים טמון ביכולתם לשמש ארגומנטים לפונקציות.



תרשים 11.11
הקצאת ערך למצביע.

מצביעים ופונקציות

ללא מצביע, הדרך היחידה להעברת ארגומנט לפונקציה היא באמצעות ערך. פירוש הדבר שהערך של הפרמטר הקורא מועבר אל הפרמטר הקולט. עותק של הערך מוצב בעמדת זיכרון נפרדת. הכנסת שינוי בערך של הפרמטר הקולט אינה משנה את ערכו של הפרמטר המשגר.

כאשר מעבירים פרמטר אל מצביע באמצעות כתובתו של הפרמטר, הוא אינו משוכפל. במקום זאת, אנו יוצרים משתנה נוסף שמצביע אל אותה עמדה בזיכרון. לפיכך, אם משנים את הערך של משתנה המצביע, משתנה גם הערך של המשתנה המשגר.

תוכנית 11.5 מזינה מחרוזת ותו. לאחר מכן התוכנית מונה את מספר הפעמים שהתו מופיע במחרוזת, ומציגה את ההופעה הראשונה של התו במחרוזת.

תוכנית 11.5: תוכנית שעושה שימוש במצביעים כדי להחזיר ערכים

```
/*letcount.c*/
main()
{
char name[20], letter;
int number, start;
puts("Enter a name\n");
gets(name);
printf("Enter a character");
letter=getchar();
countlet(name, letter, &number, &start);
printf("\nThe letter %c is in %s %d times\n",letter, name,
number);
printf("Starting position is %d", start);
}


countlet(ndplume, alpha, count, first)
char ndplume[], alpha;
int *count, *first;
{
     int index, flag;
     *count=0;
     index=0;
     flag=0;
     *first=0;
```

```
while(ndplume[index] != '\0')
{
      if(ndplume[index]== alpha)
      {
            *count=*count+1;
                  if(flag==0)
      {
                  *first=index+1;
                  flag=1;
      }
}
      index++;

}
}
```

אחרי הזנת המחרוזת והתו, אנו מעבירים ארבעה משתנים אל הפונקציה ()countlet: המחרוזת, התו, וכתובותיהם של המשתנים number ו-start, שמסומנים בתו & המופיע לפני שמותיהם.

הפונקציה מקצה את הכתובות למשתני מצביע קולטים: כתובתו של number מאוחסנת ב-count*, וכתובתו של start מאוחסנת ב-first* (תרשים 11.12). המשתנים מאותחלים בערך אפס, ולאחר מכן מופעלת לולאת while:

תרשים 11.12
העברת כתובות לפונקציה.

```
countlet(name, letter, &number, &start);
printf("\n The letter %c is in %s %d times\n",letter, name, number);
```

כתובתו של המשתנה number מאוחסנת במצביע ששמו "count"

כתובתו של המשתנה start מאוחסנת במצביע ששמו "first"

```
countlet(ndplume, alpha, count, first)
char ndplume[], alpha;
int *count, *first;
```

```
while(ndplume[index] != '\0')
```

לולאת ה-while תבדוק כל אחד מהתווים במחרוזת, ותמשיך בפעולתה כל עוד התו הנבדק שונה ממסיים האפס. כאשר היא נתקלת במסיים האפס, הלולאה מסתיימת.

מיקום הסוגריים המסולסלים בלולאת ה-while חשוב ביותר; הוראת if אחת מקננת כולה בתוך השנייה (כפי שמודגם בתרשים 11.13). תנאי ה-if הראשון בוחן אם התו הנבדק זהה לתו שברצונך למנות. במידה שכן, המונה גדל ב-1. במקרה שלנו, המונה הוא משתנה מצביע:

```
*count=*count+1;
```

פעולה זו מוסיפה 1 לערך המאוחסן בכתובת שמכיל המצביע. מכיוון שהמצביע מכיל את כתובתו של המשתנה number, המשתנה number גדל ב-1, למרות שזהו משתנה מקומי בפונקציה ()main. שים לב שלא ניתן להשתמש בצורת הכתיב:

```
*count++
```

מכיוון שהוראה כזו תשנה את הכתובת אליה מתייחס המצביע, ולא את תוכנה של אותה כתובת.

תנאי ה-if השני בוחן את ערכו של המשתנה flag. אם flag שווה אפס, זוהי ההופעה הראשונה של התו אותו מחפשים. משתנה המצביע first* מקבל את מיקומו של התו במחרוזת, פלוס 1. (הוספת 1 לאינדקס נועדה כדי שהספירה תתחיל מהתו הראשון במחרוזת, כפי שנהוג למנות, במקום מאפס). ערכו של flag נקבע ל-1, כדי שנקודת ההתחלה לא תשתנה כאשר התוכנית תאתר את התו המתאים הבא במחרוזת.

אחרי תנאי ה-if האינדקס גדל, כדי שהחזרה הבאה על הלולאה תבחן את התו הבא במחרוזת. חשוב לוודא שההוראה ++index מופיעה אחרי התוחם הסוגר את הוראת ה-if החיצונית, אך לפני זוג הסוגריים המסולסלים שמסיימים את התוכנית: תוחם אחד מסיים את הוראת ה-while, והשני מסיים את הפונקציה. אם ההוראה תוצב במקום אחר כלשהו, הפונקציה לא תפעל כהלכה.

כאשר הפונקציה מסיימת את פעולתה, השליטה חוזרת לפונקציה ()main, שמציגה את המחרוזת, התו, מספר ההופעות ונקודת ההתחלה.

שים לב שהפונקציה אינה מחזירה ארגומנט בקריאה ()return. במקום זאת, הערכים של number ושל start הוקצו באמצעות מצביעים. על ידי הקצאת ערכים למצביעים בפונקציה, התוכנית "מחזירה" את הערכים למשתנים שבארגומנט הקריאה.

ניתן לכתוב את התוכנית הזו גם ללא מצביעים, על ידי הכרזה על המשתנים number ו-start כמשתנים גלובליים. כך ניתן לאזכר אותם ב-()main וב-()countlet, במקום להעבירם כארגומנטים. עם זאת, השימוש במצביעים מעניק למשתמש בקרה רבה יותר. המשתנים נותרים אומנם משתנים מקומיים, אולם באפשרותך לבחור לשנות אותם על ידי העברת כתובותיהם.

תרשים 11.13
מיקום הסוגריים המסולסלים כדי לציין את קבוצות ההוראות.

```
countlet(ndplume, alpha, count, first)
char ndplume[], alpha;
int *count, *first;
{
    int index, flag;
    *count=0;
    index=0;
    flag=0;
    *first=0;
while(ndplume[index] != '\0')
{
    if(ndplume[index]== alpha
    {
        *count=*count+1;
    if(flag==0)
    {
        *first=index+1;
        flag=1;
    }
  }
   index++;

}
}
```

אם התו מתאים, המונה גדל

מציב את משתנה המיקום הראשון כאשר התו מתאים

בכל חזרה מתבצע הביטוי index++

כדי לעבוד עם קבצים ולשגר את הפלט למדפסת, במקום אל הצג, יהיה עליך להשתמש במצביעים. בנושאים אלה נדון בפרק 12.

שאלות

1. מתי יש להשתמש בטיפוס מבנה?
2. כיצד מכריזים על מבנה?
3. מה ההבדל בין תווית המבנה לבין משתנה המבנה?
4. כיצד מאזכרים פריט מבנה?
5. האם ניתן ליצור מערך של מבנים?
6. האם מבנה יכול להכיל פריטים המשתייכים לטיפוס נתונים אחד?
7. מהו מצביע?
8. אם תוכנית כלשהי כוללת את ההכרזה float *num, מה ההבדל בין המשתנים num ו-num*?
9. מדוע משתמשים במצביעים בתוכניות C?
10. איך מעבירים מצביע לפונקציה?

תרגילים

1. כתוב תוכנית שמזינה נתונים אודות מלאי של מוצר מסוים לתוך מבנה. הנתונים כוללים את שם המוצר, מחירו, את הכמות ואת שם הספק.
2. שנה את התוכנית שכתבת בתרגיל 1, כך שתזין את הנתונים לתוך מערך בן 20 פריטים במבנה.
3. שנה את התוכנית שכתבת בתרגיל 2, כך שתציג את שווין הכולל של המלאי.
4. כתוב תוכנית שמזינה שני משתנים מטיפוס float שיהיו מקומיים ל-()main, ואז משתמשת בפונקציה כדי לחשב את החזקה הריבועית של שני המספרים.
5. הסבר מה לקוי בתוכנית הבאה:

```
main()
{
struct CD
    {
    char description[40];
    char category[12];
```

```
    char name[20];
    float cost;
    int number;
    } disc;
    puts("Enter disk information");
    printf("Enter the name:");
    gets(name);
    printf("Enter the description:");
    gets(description);
    printf("Enter the category:");
    gets(category);
    printf("Enter the cost:");
    scanf("%f", &cost);
    printf("Enter the slot number:");
    scanf("%d", &number);
    puts("The information on the CD is:");
    printf("Name:            %s\n",name);
    printf("Description:     %s\n",description);
    printf("Category:        %s\n",category);
    printf("Cost:            %6.2f\n",cost);
    printf("Location:        %d\n",number);
}
```

12

שיגור פלט לכונן ולמדפסת

הצגת מידע על-גבי המסך היא אומנם פעולה שימושית, אך מוגבלת. גם כאשר התוכנית מאפשרת למשתמש להשהות את התצוגה די זמן כדי לקרוא את המידע, הרי שמרגע שתצוגת המסך התחלפה, יש להריץ את התוכנית פעם נוספת כדי לעיין במידע שוב.

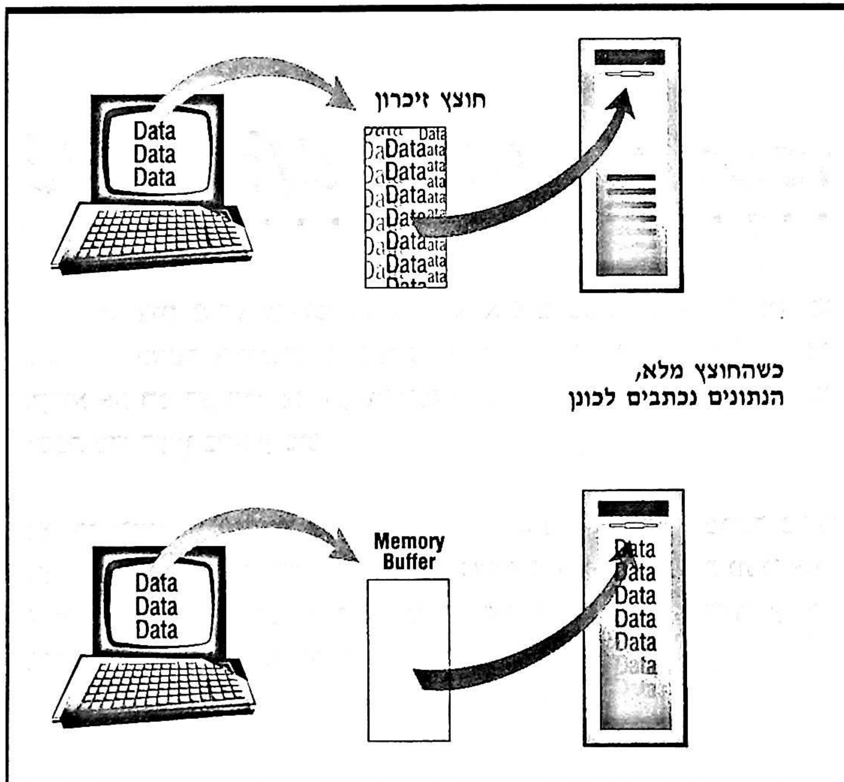
כמו כן, התוכנית שומרת על ערכיהם של משתנים אך ורק במהלך פעולתה; ברגע שעוצרים את התוכנית, הקלט נעלם. כך, לדוגמה, אם הקלדת את נתוני אוסף תקליטי הקומפקט-דיסק שברשותך לתוך מערך של משתנה מבנה, יהיה עליך להקלידם מחדש בפעם הבאה שתפעיל את המחשב.

כדי לשמור את המידע באופן תמידי, או כדי למסור עותקים של פלט התוכנית למשתמשים אחרים, יש להדפיס את הפלט על-גבי נייר. כדי שניתן יהיה להקליד את הנתונים פעם אחת בלבד ולאחזר אותם כאשר מתעורר הצורך בכך, יש לאחסן את המידע בכונן.

הבנת נושא הקבצים

פקודת פלט בתוכנית אינה משגרת את נתוני הפלט ישירות אל הכונן או אל המדפסת. הפלט מופנה לשטח אחסון זמני בזיכרון, שנקרא *חוצץ*. כאשר החוצץ מתמלא, הנתונים מועברים אל הכונן או אל המדפסת (ראה תרשים 12.1). נתוני קלט המוזנים מהכונן מופנים אף הם לחוצץ, ומאוחסנים שם עד שניתן יהיה להקצות אותם למשתנה, או להציגם על-גבי המסך.

תרשים 12.1
הנתונים מאוחסנים באופן זמני בחוצץ.

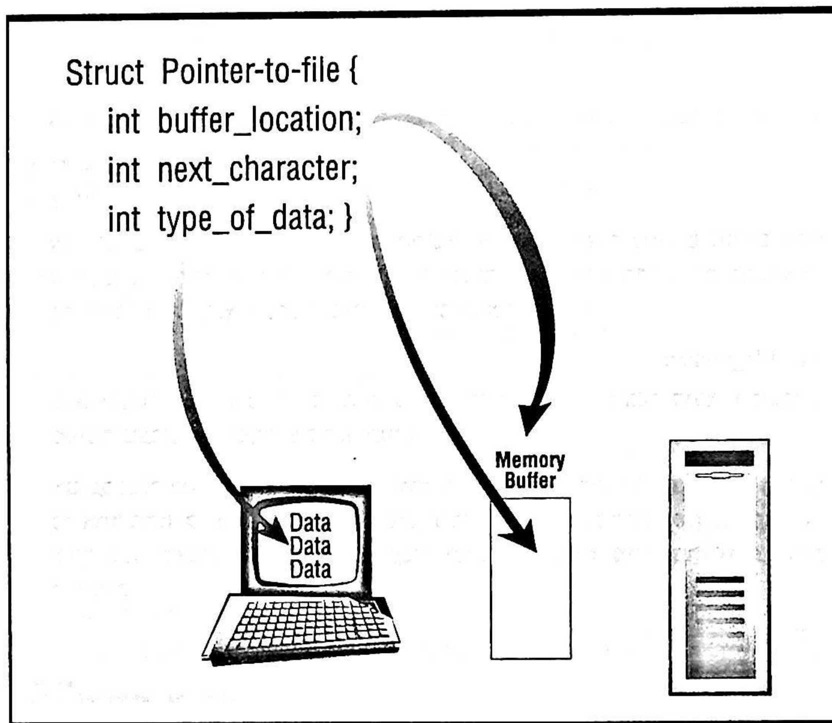


כדי להעביר נתונים אל החוצץ או ממנו, יש צורך בתקשורת בין התוכנית לבין מערכת ההפעלה של המחשב. את תפקיד הקישור ממלא הקובץ.

כאשר התוכנית יוצרת קובץ, היא מציבה בזיכרון מבנה מיוחד, שמכיל מידע הנחוץ הן לתוכנית והן למחשב כדי לקבל קלט מהקובץ או לשגר אליו פלט, או כדי להדפיס את הנתונים במדפסת.

לדוגמה, המבנה מכיל את כתובתו של חוצץ הקובץ, כדי שהמחשב יוכל לדעת היכן עליו לחפש את המידע שברצונך לשגר לכונן כפלט, או היכן להציב נתונים שברצונך להזין מהכונן כקלט לתוכנית. המבנה עוקב אחר מספר התווים שנותרו בחוצץ ואחרי מיקומו של התו הבא - שיכול להיות פלט או לאחסן קלט (ראה תרשים 12.2).

תרשים 12.2
מבנה הקובץ מאחסן מידע שנחוץ בעת הרצת התוכנית.



רוב הקומפיילרים של C ושל C++ מאחסנים את המידע אודות הקבצים בקובץ הכותרת stdio.h. קובץ זה מכיל מספר הכרזות קבועות שדרושות לצורך ביצוע פעולות בקבצים, ועשוי להכיל גם את תבנית מבנה הקובץ. כדי שניתן יהיה להשתמש בפונקציות ובפקודות של קבצים, יש להתחיל את התוכנית בפקודה הבאה:

```
#include <stdio.h>;
```

פקודה זו מאפשרת לקומפיילר גישה אל קבועי הקובץ ואל תבנית המבנה בעת קימפול וקישור התוכנית.

כאשר מזינים נתונים מקובץ שבכונן, נוצר בזיכרון המחשב עותק של הנתונים. הנתונים נותרים בכונן ללא שינוי. מסיבה זו, מכנים המתכנתים את פעולת הקלט *קריאה*. כאשר מאחסנים בכונן פלט של נתונים מהזיכרון, אנו מציבים בכונן עותק של הנתונים. פעולה זו מכונה *כתיבה*.

הערות ++C

קומפיילרים רבים של ++C כוללים קובצי כותרת נוספים המשמשים לביצוע פעולות קובץ מיוחדות. קבצים אלה עשויים להיקרא בשמות iostream.h, או fstream.h, או בשם אחר, בהתאם לתפקידם ולסוג הקומפיילר. למידע נוסף - עיין בתיעוד הנלווה לקומפיילר שברשותך.

הכרזת קובץ

אנו יוצרים קובץ כמצביע אל מבנה הקובץ שבזיכרון. כאשר כותבים נתונים בקובץ, או קוראים ממנו נתונים, התוכנית מקבלת מהמבנה את המידע הדרוש לה לביצוע הפעולה. כדי להכריז על קובץ, השתמש בצורת הכתיב הבאה:

```
FILE *file_pointer;
```

מלת המפתח FILE מורה לתוכנית שהמשתנה מצביע אל מבנה קובץ. הכוכבית יוצרת משתנה מצביע, באמצעות שם המשתנה המופיע אחריה.

אם בכוונתך להשתמש ביותר מקובץ אחד בעת ובעונה אחת, יש ליצור מצביע קובץ עבור כל אחד מהם. לדוגמה, עבור תוכנית שמעתיקה את תוכנו של קובץ אחד אל קובץ אחר, יש צורך בשני מצביעי קובץ, כמו גם עבור תוכנית שקוראת מידע מהכונן ומשגרת אותו למדפסת:

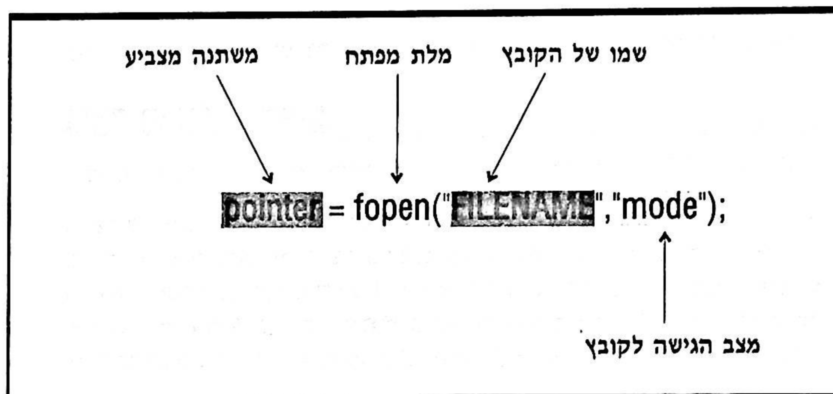
```
FILE *infile, *outfile;
```

פתיחת קובץ

הקישור בין התוכנית, הקובץ והמחשב נוצר באמצעות הפונקציה ()fopen, לפי התבנית המודגמת בתרשים 12.3.

הפונקציה מקצה את כתובתו של מבנה הקובץ למשתנה המצביע. הפרמטר כולל את שם הקובץ, שחייב להתאים למוסכמות שמות הקבצים במחשב שברשותך: ב-DOS, שם הקובץ יכול להכיל עד שמונה תווים, עם אפשרות לסיומת בת שלושה תווים. כדי להדפיס פלט, במקום לשגרו לקובץ, השתמש בשם הקובץ "PRN", במרכאות. פעולה זו משגרת את הפלט למדפסת באופן אוטומטי.

תרשים 12.3 צורת הכתיב של הפונקציה ()fopen.



ארגומנט המצב mode מציין את סוג הפעולה שבכוונתך לבצע. בשפת C ובשפת ++C יש להציב מרכאות גם סביב ארגומנט המצב, שיכול לקבל אחד הערכים הבאים:

r — מציין שבכוונתך *לקרוא* נתונים מהקובץ אל המחשב. אם הקובץ אינו קיים כבר בכונן, התוכנית תציג שגיאת הרצה.

w — מציין שבכוונתך *לכתוב* בקובץ או במדפסת. אם הקובץ עדיין אינו קיים בכונן, מערכת ההפעלה תיצור אותו. מאידך, אם הקובץ קיים, כל הנתונים הנוכחיים המאוחסנים בו יימחקו.

a — מציין שבכוונתך *להוסיף* נתונים לסופו של הקובץ. מערכת ההפעלה תיצור את הקובץ, במקרה שהקובץ אינו קיים כבר בכונן. אם הקובץ קיים, נתוני הפלט יוספו בסופו של הקובץ, מבלי למחוק את תוכנו.

לדוגמה, כדי ליצור קובץ ששמו CD.DAT שיאחסן את נתוני אוסף תקליטי הקומפקט-דיסק שלך, יש לכתוב פקודה כזו:

```
FILE *cdfile;
cdfile = fopen("CD.DAT","w");
```

אם על התוכנית לקרוא נתונים מהקובץ, במקום לכתוב לקובץ, השתמש בפורמט הבא של ()fopen:

```
FILE *cdfile;
cdfile = fopen("CD.DAT","r");
```

שים לב ששמו של הקובץ וארגומנט המצב mode נתונים במרכאות, מכיוון שהם מועברים אל הפונקציה ()fopen כמחרוזות. ניתן גם להקליד את שמו של הקובץ מהמקלדת כמשתנה מחרוזת, ואז להשתמש בשמו של המשתנה כארגומנט, ללא מרכאות.

אם ברצונך להדפיס את המידע אודות אוסף התקליטים, השתמש בפקודה כזו:

```
FILE *cdfile;
cdfile = fopen("PRN","w");
```

כאשר משגרים נתונים למדפסת, ניתן להשתמש במצב w בלבד.

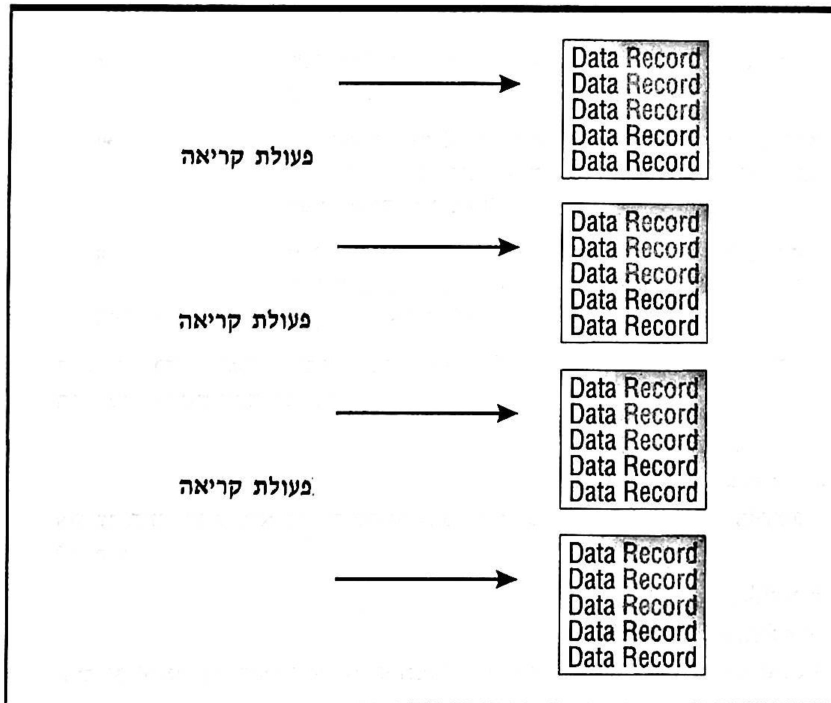
כיצד פועלים קבצים



הערה
++C וכן קומפיילרים רבים אחרים של ANSI C מאפשרים לפתוח קובץ שישמש הן לקריאה והן לכתיבה, בעת ובעונה אחת. בעבודה עם קומפיילרים אלה, ארגומנט המצב נכתב כך: +r, +w או +a.

שפת C עוקבת אחר מיקומך בקובץ באמצעות מצביע מיוחד.

כאשר קוראים נתונים מקובץ, המצביע מציין את מיקומו של הנתון הבא שייקרא מהכונן. כשהקובץ נפתח לראשונה במצב r, המצביע מוצב על התו הראשון בקובץ. אחרי כל פעולת קריאה, המצביע נע אל הנתון הבא שייקרא. גודלה של התנועה תלוי בכמות המידע שנקרא בכל פעם (תרשים 12.4). אם התוכנית קוראת תו אחד בלבד בכל פעולת קריאה, המצביע נע אל התו הבא. אם קוראים מבנה שלם, המצביע נע אל המבנה הבא. אחרי קריאת הנתון האחרון בקובץ, המצביע מועבר אל קוד מיוחד המכונה סמן-קצה-קובץ. אם תנסה לקרוא נתונים מעבר לסמן-קצה-הקובץ, תוצג שגיאת הרצה.



תרשים 12.4
המצביע עוקב אחר המיקום הנוכחי בקובץ.

גם כשפותחים קובץ במצב w המצביע מוצב בתחילת הקובץ, כך שהפלט הראשון מוחדר בתחילת הקובץ. עם סגירת הקובץ, מערכת ההפעלה מציבה את סמן-קצה-הקובץ אחרי הנתון האחרון. אם הקובץ קיים כבר בשעה שהוא נפתח במצב w, הנתונים המאוחסנים בקובץ ישוכתבו על ידי הנתונים החדשים הנכתבים בו. נתונים שלא שוכתבו יופיעו אחרי סמן-קצה-הקובץ, כדי שלא יהיו זמינים לפעולת קריאה. זו הסיבה שפתיחת קובץ קיים במצב

w מוחקת את תוכנו - גם כאשר פותחים וסוגרים את הקובץ מבלי לכתוב בו נתונים כלשהם.

כאשר פותחים קובץ במצב a, המצביע מוצב על סמן-קצה-הקובץ. הנתונים החדשים שברצונך להוסיף מוחדרים לקובץ, וסמן-קצה-הקובץ מוצב אחריהם.

הימנעות משגיאות הרצה

לעתים מערכת ההפעלה אינה יכולה לפתוח את הקובץ שצוין בפקודה ()fopen. ייתכן שלא נותר מקום בכונן, או שאתה מנסה לקרוא נתונים מקובץ שאינו קיים. הסיבה יכולה גם להיות ניסיון להדפיס בשעה שהמדפסת אינה מופעלת או כשאינה מכילה נייר.

אם ננסה להשתמש בקובץ שלא נפתח, התוכנית תיעצר בשגיאת הרצה. כדי להימנע מכך, השתמש בהוראת if שעוצרת את התוכנית במקרה שלא ניתן לפתוח את הקובץ המבוקש.

ניתן להיעזר במבנה if כדי לפתוח קובץ ולבחון אם הוא נפתח כהלכה:

```
if ((cdfile = fopen("CD.DAT","w")) == NULL)
    {
    puts("Unable to open the file");
    exit();
    }
```

NULL הוא ערך מיוחד שמוגדר בקובץ הכותרת stdio.h. אם מתגלה שגיאה בפתיחת הקובץ, מערכת המחשב מחזירה את הערך NULL במקום את כתובתו של מבנה הקובץ, והתוכנית מסתיימת.

ניתן להשתמש במבחן דומה גם לצורך שיגור פלט למדפסת:

```
if((cdfile = fopen("prn","w")) == NULL)
    {
    puts("Turn on the printer, then run this again");
    exit();
    }
```

אם המדפסת אינה מופעלת, תוצג שורת ההנחיה המורה להפעיל את המדפסת ולהריץ את התוכנית פעם נוספת. באפשרותך גם להציב את התנאי בתוך לולאת while, כדי להעניק למשתמש שהות לתקן את הבעיה:

```
while((cdfile = fopen("prn","w")) == NULL)
    {
     puts("Turn on the printer, then press Enter");
    flag = getchar();
    }
```

שים לב - שיטה זו עלולה שלא לפעול במערכות מסוימות. אם המדפסת אינה מוכנה לפעולה, מערכת ההפעלה תציג הודעה שמאפשרת למשתמש לבחור בין הפסקת הפעולה (abort) או ניסיון נוסף (retry). הפעל את המדפסת והקש R כדי לבצע ניסיון נוסף.

הערות ++C

פעולות מיוחדות של קובצי stream הכלולים בקומפיילר ++C מאפשרות לקרוא מקבצים ולכתוב לקבצים בעזרת האופרטורים << ו->>. לדוגמה, בקומפיילרים מסוימים, ניתן לפתוח קובץ לקריאה כך:

```
ifstream file_pointer(file_name)
```

וקובץ לכתיבה, כך:

```
ofstream file_pointer(file_name)
```

ייתכן שיהיה עליך לעשות שימוש בקובץ כותרת מיוחד עבור פעולות אלה. עבור שמו של קובץ הכותרת ואופן השימוש בו, עיין בתיעוד הנלווה לקומפיילר שברשותך.

סגירת קובץ

לאחר סיום ביצוע פעולות קריאה מקובץ או כתיבה לקובץ, יש לסגור את הקישור בין הקובץ לבין המחשב, כך:

```
fclose(file_pointer)
```

סגירת הקובץ מבטיחה שכל הנתונים בחוצץ נכתבו בפועל בקובץ. אם מסיימים את התוכנית קודם לסגירת הקובץ, קיים חשש שנתונים מהחוצץ שעדיין לא נכתבו, לא ייכללו בפלט ויאבדו. ללא סגירת הקובץ לא ניתן להחדיר כהלכה את סמן-קצה-הקובץ, ומערכת ההפעלה עלולה להתקשות בגישה אל הקובץ במועד מאוחר יותר.

בנוסף, סגירת הקובץ משחררת את מצביע הקובץ, כדי שניתן יהיה להשתמש בו עבור קובץ אחר, או כדי לבצע פעולה נוספת באותו קובץ. לדוגמה, נניח שברצונך ליצור קובץ, ולאחר מכן לוודא שהנתונים נקלטו כהלכה. התוכנית יכולה לעשות שימוש במבנה המתואר בתוכנית 12.1.

תוכנית 12.1: שימוש במצביע קובץ אחד לשתי פעולות

```
FILE *cdfile;
if ((cdfile = fopen("CD.DAT","w")) == NULL)
     {
     puts("Unable to create the file");
     exit();
     }

Instructions for writing to the file

fclose(cdfile);
if ((cdfile = fopen("CD.DAT","r")) == NULL)
     {
     puts("Unable to open the file");
     exit();
     }

Instruction from reading from the file

fclose(cdfile);
```

תחילה, הקובץ נפתח במצב w והנתונים נכתבים לתוכו. הקובץ נסגר, ונפתח פעם נוספת במצב r, כדי שניתן יהיה לקרוא את הנתונים ולהציגם על המסך.

קומפיילרים מסוימים מאפשרים למשתמש לוודא שכל הנתונים נכתבו לקובץ על ידי "ניקוי" החוצץ באמצעות הפקודה:

flush()

פקודה זו גורמת לכתיבה לכונן, או להדפסה, של כל הנתונים בחוצץ, גם במקרה שהקובץ לא נסגר.

פונקציות קלט ופלט

ישנן מספר דרכים להעביר נתונים מקבצים ואל קבצים:

- לכתוב את הנתונים לקובץ או לשגרם למדפסת תו אחר תו, בעזרת הפונקציות ()putc או ()fputc.
- לקרוא נתונים מקובץ תו אחר תו, בעזרת ()getc או ()fgetc.

- לכתוב נתונים לקובץ או לשגרם למדפסת כמחרוזות (שורה אחר שורה), בעזרת הפונקציה ()fputs.
- לקרוא נתונים מקובץ כמחרוזת (שורה אחר שורה), בעזרת הפונקציה ()fgets.
- לכתוב תווים מפורמטים, מחרוזות ומספרים לכונן, או לשגרם למדפסת, בעזרת הפונקציה ()fprintf.
- לקרוא תווים מפורמטים, מחרוזות ומספרים מקובץ בעזרת הפונקציה ()fscanf.
- לכתוב מבנה שלם באמצעות הפונקציה ()fwrite.
- לקרוא מבנה שלם באמצעות הפונקציה ()fread.

עבודה עם תווים

העברת נתונים בשיטת תו-אחר-תו היא פעולת הקובץ הבסיסית ביותר. אומנם אין זו השיטה היעילה ביותר לטיפול בנתונים, אך ניתן להדגים בעזרתה את אופן השימוש בקבצים. התוכנית הבאה, לדוגמה, כותבת תווים לקובץ, עד שהמשתמש מקיש Enter:

```
/*fputc.c*/
#include "stdio.h"
main()
{
        FILE *fp;
        char letter;
        if((fp = fopen("MYFILE","w"))==NULL)
        {
                puts("Cannot open the file");
                exit();
        }
        do
        {
                letter=getchar();
                fputc(letter, fp);
        }
        while(letter!='\r');
        fclose(fp);
}
```

הקובץ נפתח במצב w - ובמקרה שהקובץ MYFILE אינו קיים, התוכנית יוצרת אותו. לולאת ה-do מזינה סדרה של תווים באמצעות הפונקציה ()getchar, וכותבת אותם לקובץ בעזרת הפונקציה ()putc. צורת הכתיב של הפונקציה ()putc היא:

```
putc(char_variable, file_pointer);
```

ניתן גם להשתמש בהוראת ()fputc עם אותם ארגומנטים.

הלולאה חוזרת על עצמה עד שהמשתמש מקיש Enter. הקשת Enter מזינה את תו החזרה (r\) והקובץ נסגר.

קלט ()getchar באמצעות חוצץ

בעבודה עם קומפיילרים שמשתמשים בחוצץ בעת הזנת נתונים עם הפונקציה ()getchar, השתמש בפונקציה ()getche כדי להזין תווים בודדים לצורך כתיבה לקובץ בכונן. החלף את מבנה ה-do...while בלולאה הבאה:

```
while((letter=getche())!='\r')
        fputc(letter, fp);
```

קריאת תווים מקובץ

כדי לקרוא תו מקובץ, השתמש בפונקציה ()getc או בפונקציה ()fgetc, שצורת הכתיב שלה היא:

```
char_variable = getc(file_pointer);
```

לדוגמה, כדי להזין תו מקובץ שעליו מצביע המצביע fp, השתמש בהוראה:

```
letter = getc(fp);
```

התוכנית הבאה קוראת את הקובץ שיצרנו בדוגמה הקודמת:

```
/*fgetc.c*/
#include "stdio.h"
main()
{
    FILE *fp;
    int letter;
    if((fp = fopen("MYFILE","r"))==NULL)
    {
```

```
            puts("Cannot open the file");
            exit();
        }
        while ((letter=fgetc(fp)) != EOF)
            printf("%c",letter);
        fclose(fp);
    }
```

אנו פותחים את הקובץ במצב r וקוראים את התווים בלולאת ה-while. מרבית העבודה מתבצעת בתוך הפרמטר של הוראת ה-while. ההוראה:

```
letter=fgetc(fp)
```

מקצה תו מהקובץ למשתנה letter. הלולאה חוזרת על עצמה כל עוד המשתנה אינו שווה EOF, שהוא קבוע מיוחד המייצג את קצה-הקובץ. כאשר המצביע מגיע לקצה הקובץ, הערך המוחזר למשתנה letter יהיה שווה לערך EOF, המוכרז בקובץ הכותרת stdio.h.

הדוגמאות שהצגנו עושות שימוש בשמות זהים של מצביע הקובץ והמשתנה, הן לצורך כתיבת נתונים לקובץ והן לצורך קריאה מהקובץ. אך הדבר אינו הכרחי. באפשרותך לכתוב לקבצים ולקרוא מהם, תוך שימוש בשמות שונים למצביעים ולמשתנים, כל עוד שם הקובץ זהה עבור שתי הפעולות.

עבודה עם שורות

במקום לעבוד עם תווים בודדים, ניתן לקרוא ולכתוב שורת מלל שלמה - מחרוזת - בעזרת הפונקציות ()fputs ו-()fgets.

מבנה הפונקציה ()fputs הוא:

```
fputs(string_variable, file_pointer);
```

הפונקציה כותבת לקובץ או משגרת למדפסת מחרוזת שלמה, אך אינה מוסיפה פקודת שורה-חדשה לסופה של המחרוזת. אם ברצונך לאחסן בקובץ כל שורה בנפרד, או להדפיס כל מחרוזת בשורה נפרדת, עליך להוסיף את פקודת השורה-החדשה בעצמך. לדוגמה, התוכנית הבאה יוצרת קובץ של שמות:

```
/*fputs.c*/
#include "stdio.h"
main()
{
FILE *fp;
```

```
char flag;
char name[20];
if((fp = fopen("MYFILE","w"))==NULL)
    {
    puts("Cannot open the file");
    exit();
    }
flag = 'y';
while(flag!='n')
{
    puts("Enter a name");
    gets(name);
    fputs(name, fp);
    fputs("\n",fp);
    printf("Do you have another name?");
    flag=getchar();
    putchar('\n');
}
fclose(fp);
}
```

לולאת ה-while חוזרת על עצמה עד שהמשתמש מקיש את האות n בתגובה לשאלה. הלולאה מזינה שם באמצעות הפונקציה ()gets; לאחר מכן נכתב השם לקובץ בעזרת הפקודה ()fputs. בשלב זה נוצר פלט של קוד שורה-חדשה והתוכנית שואלת את המשתמש אם ברצונו להזין שם נוסף.

אם הקומפיילר שברשותך כולל את הפונקציה ()strlen, ניתן לפשט את החזרות, כך:

```
printf("Please enter a name: ");
gets(name);
while(strlen(name) > 0)
{
    fputs(name, fp);
    fputs("\n",fp);
    printf("Please enter a name: ");
    gets(name);
}
```

המחרוזת שהמשתמש מקליד מוקצית למשתנה name. התוכנית משווה את אורכה של המחרוזת ל-0. אם המשתמש מקיש Enter מבלי להקיש שם, אורכה של המחרוזת יהיה 0, והלולאה תסתיים. אם המשתמש מקליד תו אחד לפחות, המחרוזת וקוד שורה-חדשה נכתבים לכונן.

בקומפיילרים מסוימים ניתן לקצר את האלגוריתם אף יותר, כך:

```
printf("Please enter a name: ");
while(strlen(gets(name)) > 0)
{
      fputs(name, fp);
      fputs("\n",fp);
      printf("Please enter a name: ");
}
```

קלט המחרוזת מבוצע בתוך תנאי ה-while עצמו.

על ידי שימוש בשם הקובץ "prn" ניתן להדפיס את המחרוזות, במקום לשמור אותן בכונן. פתח את הקובץ בעזרת הפקודה:

```
if((fp = fopen("prn","w"))==NULL)
```

כדי ליצור תוכנית הדפסה עליך להכריז על המחרוזת בתחום 81 תווים, כדי שניתן יהיה להציג שורה מלאה על המסך קודם להקשת Enter. תוכנית 12.2 מדגימה כיצד ניתן לבנות מעבד תמלילים פשוט. מכיוון שהשורה אינה משוגרת למדפסת עד להקשת Enter, ניתן להשתמש במקש Backspace לתיקון שגיאות בשורה בזמן ההקלדה.

תוכנית 12.2: תוכנית להדפסת שורות פלט

```
/*wp.c*/
#include "stdio.h"
main()
{
FILE *fp;
char line[81];
if((fp = fopen("prn","w"))==NULL)
     {
     puts("Cannot access the printer");
     exit();
     }
puts("Type your lines of text, pressing Enter after each\n");
puts("Press Enter on a new line to stop\n");
```

```
gets(line);
while(strlen(line) > 0)
{
     fputs(line, fp);
     fputs("\n",fp);
     gets(line);
}
fclose(fp);
}
```

קריאת מחרוזות

קריאת מחרוזת מקובץ מתבצעת באמצעות הפונקציה ()fgets, לפי הפורמט הבא:

```
fgets(string_variable, length, file_pointer);
```

הפונקציה מזינה שורה שלמה, עד לקוד השורה-החדשה, אולם לא יותר מ-length פחות תו אחד. הפרמטר length הוא מספר שלם, או משתנה או קבוע מטיפוס int, שמבטא את המספר המירבי של תווים שניתן להזין.

להלן תוכנית שקוראת בחזרה את סדרת השמות שיצרנו בדוגמה הקודמת:

```
/*fgets.c*/
#include "stdio.h"
main()
{
FILE *fp;
char name[12];
if((fp = fopen("MYFILE","r"))==NULL)
    {
    puts("Cannot open the file");
    exit();
    }
 while(fgets(name,12,fp)!=NULL)
 {
     printf(name);
 }
 fclose(fp);
```

}

הקלט מתבצע בפרמטר של ביטוי ה-while כל עוד הערך שהתוכנית קוראת שונה מ-NULL. בעת קריאת מחרוזות, כאשר המצביע מגיע לסופו של הקובץ, מוקצה הערך NULL למשתנה המחרוזת. לכן, יש להשתמש תמיד ב-NULL כדי לאתר את סופו של הקובץ בקריאת מחרוזות; בעת קריאת תווים, השתמש ב-EOF.

אם ברצונך לכתוב תוכנית כללית לקריאת קובץ מלל כלשהו, קבע את גודלו של הארגומנט length ל-80.

הפקודה ()printf, דרך-אגב, משמשת כאן ליצירת פלט של תוכן משתנה המחרוזת ללא מציין פורמט. כל מחרוזת שנקראת מהקובץ כוללת פקודת שורה-חדשה שנכתבה לקובץ על ידי ההוראה fputs("\n",fp). אין צורך בפקודת שורה-חדשה נוספת בתוך הפונקציה printf().

קלט ופלט מפורמטים

פקודות קובץ שמטפלות בתווים ובמחרוזות יכולות אך ורק לקרוא מלל ולכתבו. אם ברצונך ליצור קובץ או תדפיס שמכילים ערכים מספריים, עליך להשתמש בפונקציות ()fprintf ו-()fscanf. צורת הכתיב של פקודות אלה דומה לזו של ()printf ו-()scanf, אולם הן כוללות גם את מצביע הקובץ שמציין מהיכן יש לקרוא את הנתונים, או להיכן לכתוב אותם.

```
fprintf(file_pointer, control_string, data_list);
fscanf(file_pointer, control_string, data_list);
```

תוכנית 12.3 מזינה נתוני מלאי וכותבת אותם לקובץ. הקלט הראשון מבוצע באמצעות פקודת ()gets, לפני תחילתה של לולאת ה-while. הלולאה חוזרת כל עוד אורכו של כל אחד משמות הפריטים הוא תו אחד לפחות. הלולאה מסתיימת כאשר מקישים Enter במקום שם.

תוכנית 12.3: תיעוד פלט מפורמט

```
/*fprintf.c*/
#include "stdio.h"
main()
{
FILE *fp;
char name[20];
int  quantity;
float  cost;
if((fp = fopen("MYFILE","w"))==NULL)
     {
     puts("Cannot open the file");
```

```
        exit();
        }
printf("Please enter the item name: ");
gets(name);
while(strlen(name) > 0)
{
        printf("Please enter the cost: ");
        scanf("%f", &cost);
        printf("Please enter the quantity: ");
        scanf("%d", &quantity);
        fprintf(fp, "%s %f %d\n", name, cost, quantity);
        printf("Please enter a name: ");
        gets(name);

}
fclose(fp);
}
```

שים לב שהשם השני מוזן בתור השורה האחרונה של הלולאה. מבנה זה מאפשר למשתמש לסיים את הלולאה על ידי הקשת Enter פעם אחת. מתכנתים מתחילים עשויים לכתוב את הלולאה כך:

```
do
{
        printf("Please enter a name: ");
        gets(name);
        printf("Please enter the cost: ");
        scanf("%f", &cost);
        printf("Please enter the quantity: ");
        scanf("%d", &quantity);
        fprintf(fp, "%s %f %d\n", name, cost, quantity);
}
while(strlen(name) > 0)
```

גם כך הלולאה תפעל כשורה, אולם כדי לסיים את הלולאה על המשתמש להקיש Enter שלוש פעמים: פעם אחת עבור השם, ופעמיים נוספות כדי לעקוף את ההנחיות עבור cost ועבור quantity.

בתוך לולאת ה-while, המחיר והכמות (quantity, cost) של כל פריט מוזנים על ידי שימוש בפונקציות ()scanf, ולאחר מכן הנתונים נכתבים לקובץ על הדיסק בפקודה אחת:

```
fprintf(fp, "%s %f %d\n", name, cost, quantity);
```

שים לב שבסופה של כל שורה נכתב לקובץ קוד שורה-חדשה. אם נסקור את הקובץ באמצעות פקודת TYPE של DOS, תופיע כל שורת נתונים בנפרד:

```
disk 1.120000 100
ribbon 7.340000 150
toner 75.000000 3
```

ללא פקודת השורה-החדשה, המלל היה מופיע ברצף אחד, כך:

```
disk 1.120000 100ribbon 7.340000 150toner 75.000000 3
```

שים לב שאין רווח בין ערך הכמות של פריט אחד, לבין שמו של הפריט הבא אחריו. עם זאת, ניתן יהיה לקרוא את הקובץ ללא קושי, מכיוון שהקומפיילר מזהה היכן מסתיים ערך מספרי ומתחילה מחרוזת.

לעומת זאת, נניח שהערך הראשון והאחרון של כל פריט היו מחרוזות, כך:

```
disk 1.120000 Memoryexribbon 7.340000 Okaydatatoner 75.000000 HP
```

כשהתוכנית תקרא את המחרוזת בסופו של הפריט הראשון, היא תקרא גם את המחרוזת שפותחת את הפריט השני. לדוגמה, הפריט הראשון יקרא כך:

```
disk 1.120000 Memoryexribbon
```

כל נתוני הפלט, אפילו ערכים מטיפוס int ו-float, מאוחסנים על-גבי הדיסק כתווי מלל. בהמשך נדון בנושא זה בפירוט.

קריאת קבצים מפורמטים

קלט מקובץ מפורמט מתבצע באמצעות הפונקציה ()fscanf. הפונקציה ()fscanf פועלת באופן דומה לפונקציה ()scanf, אלא שהיא מקבלת את הקלט מקובץ ולא מהמקלדת. לרוע המזל, ()fscanf סובלת מחסרונות דומים לאלה של ()scanf. היא אינה יכולה לקרוא מחרוזות שכוללות רווחים, ומחזירה תוצאות שגויות במקרה שהנתונים בשטף אינם תואמים לטיפוסי הנתונים הצפויים במחרוזת הבקרה.

תוכנית 12.4 קוראת את קובץ המלאי המפורמט. שים לב שלולאת ה-while חוזרת כל עוד הקלט שונה מ-EOF (קצה הקובץ).

תוכנית 12.4: קריאת קובץ מלל מפורמט

```
/*fscanf.c*/
#include "stdio.h"
main()
{
FILE *fp;
char name[20];
int  quantity;
float  cost;
if((fp = fopen("MYFILE","r"))==NULL)
     {
     puts("Cannot open the file");
     exit();
     }
while(fscanf(fp,"%s %f %d", name, &cost, &quantity) != EOF)
{
     printf("Item: %s\n", name);
     printf("Cost: %.2f\n", cost);
     printf("Cost: %d\n", quantity);
}
fclose(fp);
}
```

עבודה עם מבנים

אחת הדרכים להתגבר על חסרונותיה של הפונקציה ()fscanf היא לשלב אלמנטים של נתונים במבנה, ואז ליצור פלט וקלט של המבנה כולו בעת ובעונה אחת. ניתן לכתוב מבנה לדיסק באמצעות הפונקציה ()fwrite, ולקרוא מבנה מהדיסק באמצעות הפונקציה ()fread.

צורת הכתיב של ()fwrite היא כדלקמן:

```
fwrite(&structure_variable, structure_size,
number_of_structures, file_pointer);
```

צורת כתיב זו נראית אולי מסובכת, אך למעשה קל מאד ליישמה:

- &structure_variable הוא משתנה המבנה עם אופרטור הכתובת. משתנה זה מורה לקומפיילר מהי הכתובת ההתחלתית של הנתונים שברצונך לכתוב לדיסק.
- structure_size הוא מספר התווים במבנה. במקום לספור את התווים בעצמך, השתמש בפונקציית הספרייה ()sizeof, כך:

```
sizeof(structure_variable)
```



פעולה זו מחשבת את גודלו של המבנה באופן אוטומטי.

- number_of_structures הוא מספר שלם שמציין כמה מבנים ברצונך לכתוב בעת ובעונה אחת. השתמש תמיד במספר 1, אלא אם ברצונך ליצור מערך של מבנים ולכתוב את המערך כולו כיחידה גדולה אחת.
- file_pointer הוא המצביע המסמן את הקובץ.

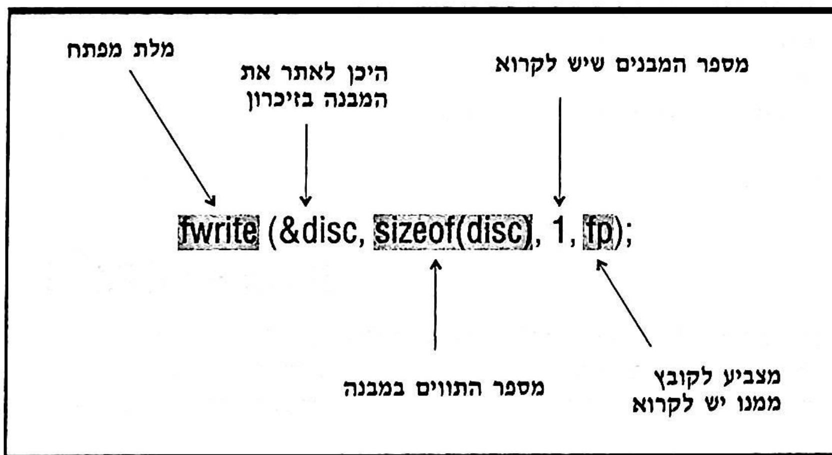
לדוגמה, נניח שברצונך לשמור את נתוני אוסף תקליטי הקומפקט-דיסק שלך על-גבי כונן. אם נשתמש במבנה CD שתיארנו בפרק 11, הפקודה לכתוב מבנה תיראה כך:

```
fwrite(&disc, sizeof(disc), 1, fp);
```

ההוראה מודגמת בתרשים 12.5.

תוכנית 12.5 מזינה נתונים לתוך המבנה CD ושומרת אותם על-גבי הכונן. התוכנית עושה שימוש בפונקציה ()gets לצורך הזנת שם הקובץ שברצונך ליצור. המשתנה שמאחסן את שם הקובץ משמש לאחר מכן בפקודה ()fopen, לפתיחת הקובץ.

תרשים 12.5
הכתיב של הפקודה ()fwrite עבור המבנה CD.



הנתונים על כל תקליט קומפקט-דיסק מוזנים מהמקלדת, ולאחר מכן נכתב המבנה כולו לכונן.

תוכנית 12.5: תיעוד המבנה CD

```
/*fwrite.c*/
#include "stdio.h"
main()
{
     FILE *fp;
     struct CD
```

```
{
char  name[20];
char  description[40];
char  category[12];
float cost;
int  number;
} disc;
char filename[25];
printf("Enter the file you want to create:  ");
gets(filename);
if((fp = fopen(filename,"w")) == NULL)
{
     printf("Can't open the file %s\n",filename);
     exit();
}
puts("Enter disk information\n");
printf("Please enter the name of the CD: ");
gets(disc.name);
while(strlen(disc.name) > 0)
{
printf("Enter the description: ");
gets(disc.description);
printf("Enter the category: ");
gets(disc.category);
printf("Enter the cost: ");
scanf("%f", &disc.cost);
printf("Enter the slot number: ");
scanf("%d", &disc.number);
fwrite(&disc, sizeof(disc), 1, fp);
printf("Please enter the name: ");
gets(disc.name);
}
fclose(fp);
}
```

קריאת מבנים

כדי לקרוא מבנה שלם בבת אחת, השתמש בפונקציה ()fread, לפי צורת הכתיב הבאה:

fread(&structure_variable, structure_size,

number_of_structures, file_pointer);

צורת הכתיב של ()fread זהה לזו של ()fwrite, למעט שם הפונקציה. תוכנית 12.6 קוראת את מבנה קובץ תקליטי הקומפקט-דיסק. הנתונים נקראים בעזרת לולאת while:

while(fread(&disc, sizeof(disc), 1, fp)==1)

הפונקציה ()fread מחזירה את מספרם של המבנים השלמים שנקראו בהצלחה. מכיוון שהארגומנט מתייחס לקריאת מבנה אחד בלבד בכל פעם, הערך שיוחזר אמור להיות 1. לולאת ה-while חוזרת כל עוד ניתן לקרוא מבנה שלם מהכונן. במקרה שלא ניתן לקרוא מבנה - מכיוון שהפונקציה הגיעה לקצה הקובץ - הפונקציה מחזירה אפס והלולאה מסתיימת.

תוכנית 12.6: קריאת המבנה CD מהכונן

```
/*fread.c*/
#include "stdio.h"
main()
{
     FILE *fp;
     struct CD
     {
     char  name[20];
     char  description[40];
     char  category[12];
     float cost;
     int  number;
     } disc;
     char filename[25];
     printf("Enter the filename to read: ");
     gets(filename);
     if((fp = fopen(filename,"r")) == NULL) {
          printf("Can't open the file %s\n",filename);
          exit();
     }
     while(fread(&disc, sizeof(disc), 1, fp)==1)
     {
```

```
        puts(disc.name);
        putchar('\n');
        puts(disc.description);
        putchar('\n');
        puts(disc.category);
        putchar('\n');
        printf("%f\n", disc.cost);
        printf("%d\n", disc.number);
        }
        fclose(fp);
}
```

טבלה 12.1 מסכמת את שיטות הקלט והפלט, וכן את המבחן בו עושה שימוש כל אחת מהפונקציות כדי לעצור את קריאת הנתונים.

טבלה 12.1
סיכום פונקציות קלט ופלט של קבצים.

characters	*putc(), fputc()*	*getc(), fgetc()*	*while != EOF*
lines	*fputs()*	*fgets()*	*while != NULL*
formatted	*fprintf()*	*fscanf()*	*while !=EOF*
structures	*fwrite()*	*fread()*	*while !=1*

קריאה לתוך מערך

התוכנית בה השתמשנו כאן להדגמת הקריאה מהכונן מסתפקת בהצגת נתוני הקלט על-גבי המסך. עם זאת, מרגע שהנתונים נקראו לתוך משתנה, ניתן לבצע עליהם כל פעולה שהיא, כמו גם להשתמש בנתונים לטעינת מערך.

תוכנית 12.7, לדוגמה, קוראת את הנתונים מקובצי הקומפקט-דיסק לתוך מערך של מבנה ה-CD (בהנחה שאין יותר מ-20 תקליטים). התוכנית עושה שימוש במספר הסידורי בכל אחד מהמיקומים של המבנה ששמו disc, כך שכל מבנה מוזן לתוך אלמנט אחר במערך. לאחר קריאתו והצגתו של כל מבנה, מחושב ערכו הכולל של האוסף, והמספר הסידורי ומשתנה המנייה גדלים בעזרת ההוראות הבאות:

```
total = total + disc[index].cost;
index++;
count++;
```

אם היינו מעוניינים אך ורק בנתוני הסכום והמנייה, היינו יכולים לקרוא כל רשומה לתוך מבנה שאינו מערך, ולעקוב באופן שוטף אחר הסכום והמנייה המצטברים. אולם, מרגע שקראנו את הנתונים לתוך מערך, ניתן להשתמש במערך לאיתור תקליט קומפקט-דיסק מסוים, או להדפיס דו"ח כלשהו.

שים לב שהתוכנית מבקשת שוב ושוב שם קובץ, עד שניתן לפתוח קובץ.

תוכנית 12.7: קריאת המבנה לתוך מערך

```
/*rarray.c*/
#include "stdio.h"
main()

{
     FILE *fp;
     struct CD
     {
     char  name[20];
     char  description[40];
     char  category[12];
     float cost;
     int  number;
     } disc[20];
     int index, count;
     float total;
     count = 0;
     total = 0;
     char filename[25];
     printf("Enter the input filename? ");
     gets(filename);
     while((fp = fopen(filename,"r")) == NULL)
      {
     printf("Can't open the file %s\n",filename);
     printf("Enter the input filename? ");
     gets(filename);
```

```
        }
        index = 0;
        while(fread(&disc[index], sizeof(disc[index]), 1, fp)==1)
        {
        puts(disc[index].name);
        putchar('\n');
        puts(disc[index].description);
        putchar('\n');
        puts(disc[index].category);
        putchar('\n');
        printf("%f\n", disc[index].cost);
        printf("%d\n", disc[index].number);
        total = total + disc[index].cost;
        index++;
        count++;

          }
          fclose(fp);
          printf("The total worth of my collection is %.2f\n", total);
          printf("I have %d CDs in my collection\n", count);

}
```

הערות ++C

בעבודה עם קומפיילרים של ++C ניתן לקרוא ולכתוב קבצים שנפתחו בעזרת הפונקציות ifstream ו-ofstream, תוך שימוש באופרטורים >> ו-<<. כדי לקרוא קובץ, השתמש בצורת הכתיבה הבאה:

```
file_pointer >> variable;
```

כדי לכתוב קובץ, השתמש בצורת הכתיבה הבאה:

```
file_pointer << variable;
```

הוספת נתונים לקובץ

כאשר פותחים קובץ קיים במצב w, נמחקים כל הנתונים בקובץ. כדי להוסיף נתונים לקובץ, יש לפתוח אותו במצב a. למעשה, במרבית הקומפיילרים ניתן להשתמש בתוכנית אחת הן ליצירת קובץ והן להוספת רשומות לקובץ. השימוש במצב a מבטיח שהקובץ ייווצר, אם אינו קיים עדיין. אם הקובץ כבר נמצא על-גבי הכונן, הנתונים החדשים יתווספו אליו.

כאשר מוסיפים נתונים לקובץ, יש להקפיד שפורמט הנתונים החדשים זהה לפורמט הנתונים השמורים בקובץ. לדוגמה, ניתן לפתוח קובץ שנוצר כסדרה של תווים, ואז לכתוב לתוכו מבנה. אולם כשתנסה לקרוא מהקובץ, תתקבל שגיאת הרצה או שיתקבלו נתונים מוזרים.

פורמט של מלל ופורמט בינארי

הפונקציות ()putc, ()fputc ו-()fputs יוצרות פלט של מלל. אם נתבונן בקובץ בעזרת פקודת TYPE, נראה את התווים בדיוק כפי שהקלדנו אותם. ניתן להשתמש בכל אחת מפונקציות אלה כדי ליצור קובץ, ואז לקרוא אותו בעזרת ()getc, ()fgetc או ()fgets. פונקציות התווים לדוגמה, יקראו את הקובץ תו-אחר-תו, אפילו אם במקור הקובץ נכתב כמחרוזות, בעזרת ()fputs. באופן דומה, פונקציות המחרוזות יקראו את הקובץ שורה-אחר-שורה, גם אם הקובץ נכתב כתווים בודדים.

תוכנית 12.8, לדוגמה, מעתיקה קובץ אחד לקובץ אחר, ומציגה את תוכנו של הקובץ על-גבי המסך. התוכנית תפעל עם כל קובץ שהוא, ללא תלות באופן בו הוא נוצר. אנו מכריזים על שני מצביעי קובץ, מכיוון שעלינו ליצור גישה לשני קבצים בעת ובעונה אחת.

תוכנית 12.8: תוכנית העתקת-קבצים

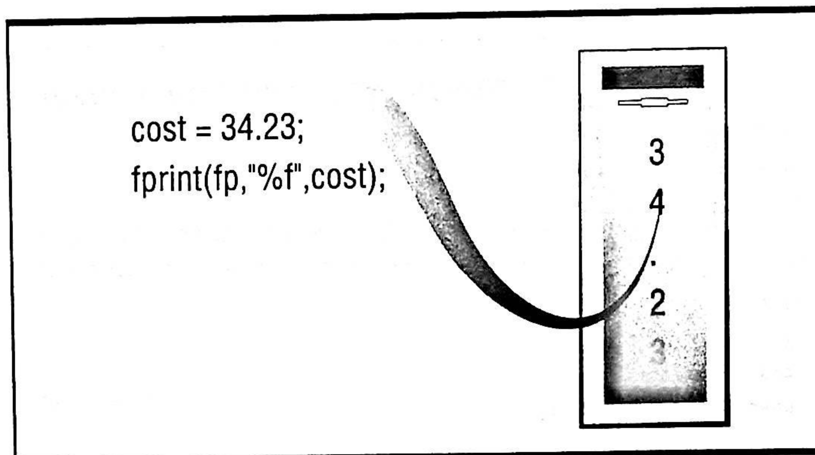
```
/*filecopy.c*/
#include "stdio.h"
main()
{
     FILE *fp1, *fp2;
     char infile[25], outfile[25];
     int letter;
     printf("Enter the filename to read: ");
     gets(infile);
     if((fp1 = fopen(infile,"r")) == NULL)
      {
          printf("Can't open the file %s\n",infile);
          exit();
      }
     printf("Enter the filename to write: ");
     gets(outfile);
     if((fp2 = fopen(outfile,"w")) == NULL)
```

```
{
    printf("Can't open the file %s\n", outfile);
    fclose(fp1);
    exit();
}
while((letter= fgetc(fp1)) != EOF)
    {
    putchar(letter);
    fputc(letter, fp2);
    }
fclose(fp1);
fclose(fp2);

}
```

הקובץ הראשון נפתח במצב r, כדי שניתן יהיה להזין את הנתונים שהוא מכיל. אם לא ניתן לפתוח את הקובץ, התוכנית מסתיימת. הקובץ השני נפתח במצב w, כדי שניתן יהיה ליצור פלט של נתונים לתוכו. אם לא ניתן לפתוח את קובץ הפלט, נסגר תחילה קובץ הקלט והתוכנית מסתיימת. מבנה זה מבטיח שקובץ הקלט שנפתח בהצלחה, לא ייפגם עם סיום התוכנית.

גם הפונקציה fprintf() כותבת את הנתונים כמלל. לדוגמה, אם נשתמש ב-fprintf() כדי לכתוב את המספר 34.23 לכונן, ייכתבו למעשה חמישה תווים, כפי שמודגם בתרשים 12.6. כאשר משתמשים ב-fscanf() לקריאת הקובץ, התווים מומרים לערכים מספריים בעת החדרתם לכתובת המשתנה.



תרשים 12.6
fprintf() מאחסנת ערכים מספריים כתווי מלל.

מכיוון ש-()fprintf מאחסנת את כל הנתונים כמלל, באפשרותך גם לקרוא את הקובץ בעזרת הפונקציות ()getc, ()fgetc או ()fgets. עם זאת, פונקציות אלה יתייחסו לנתונים המספריים כאל תווי מלל. בשימוש בפונקציה ()fgets, לדוגמה, המספרים ייקראו כתווים שמהווים חלק משורה. ניתן אומנם להציג או להדפיס את הקובץ אחרי קריאתו באמצעות ()fgets או ()fgetc, אולם לא ניתן לבצע פעולות כלשהן על פריטי הנתונים השונים.



טיפ

קומפיילרים של ++C וכן קומפיילרים רבים של ANSI C מאפשרים לך ליצור קבצים בפורמט בינארי לצורך אחסון נתונים מספריים. יש ליצור את הקובץ במצב wb, ולקרוא אותו במצב rb. התו b מסמן פורמט בינארי. לאחר מכן ניתן לכתוב ערכי מספרים שלמים בעזרת הפונקציה ()putw, ולקרוא אותם בעזרת ()getw.

פורמט בינארי

השתמש בפונקציה ()fwrite כדי לשמור משתנים מספריים בפורמט בינארי. כמות המקום שהמספרים תופסים בכונן זהה לזו שהם תופסים בזיכרון. אם נתבונן בקובץ בעזרת הפקודה TYPE, נראה תווים או סימנים חסרי משמעות במקום נתונים מספריים. סימנים אלה מייצגים את תווי ה-ASCII המקבילים לערכים המאוחסנים.

כדי לקרוא קובץ שנשמר בעזרת ()fwrite, יש להשתמש ב-()fread ולהזין את הנתונים לאותה תבנית מבנה. שמו של המבנה יכול להיות שונה מזה ששימש לכתיבת הקובץ, וכך גם שמות האלמנטים החברים במבנה, אולם סדר האלמנטים, הטיפוס והגודל של האלמנטים חייבים להיות זהים.

הדפסת נתונים

מבחינה טכנית, ניתן להשתמש בכל פונקצית פלט קובץ לצורך שיגור נתונים למדפסת: לפי תו, שורה, שורה מפורמטת או מבנה. כל שעליך לעשות הוא לפתוח את הקובץ תוך שימוש בשם "prn" ובמצב w.

עם זאת, אין זה מעשי להדפיס מבנה בעזרת הפקודה ()fwrite. ערכים מספריים יודפסו כנתונים בינאריים, ויופיעו בהדפסה כסימנים ותווים חסרי משמעות. במקום זאת, יש להדפיס נתונים במבנה בעזרת הפונקציה ()fprintf, כפי שמודגם בתוכנית 12.9. תוכנית זו פותחת שני קבצים: קובץ הכונן נפתח לצורך קריאה, וקובץ המדפסת נפתח לצורך כתיבה.

תוכנית 12.9: קריאת קובץ כונן והדפסתו

```
/*fread1.c*/
#include "stdio.h"
main()
{
     FILE *fp, *ptr;
     struct CD
     {
     char  name[20];
     char  description[40];
     char  category[12];
     float cost;
```

```
int  number;
} disc;
char filename[25];
printf("Enter the filename to read: ");
gets(filename);
if((fp = fopen(filename,"r")) == NULL)
 {
     printf("Can't open the file %s\n",filename);
     exit();
 }
if((ptr = fopen("PRN","w")) == NULL)
 {
     printf("Something is wrong with the printer\n");
     fclose(fp);
     exit();
 }
while(fread(&disc, sizeof(disc), 1, fp)==1)
{
fprintf(ptr,"CD Name          %s\n",disc.name);
fprintf(ptr,"Description:     %s\n",disc.description);
fprintf(ptr,"Category:        %s\n",disc.category);
fprintf(ptr,"Cost:            %6.2f\n",disc.cost);
fprintf(ptr,"Location:        %d\n",disc.number);
fprintf(ptr,"\n\n");
}
fclose(ptr);
fclose(fp);
}
```

כל אחד מהמבנים מוזן באמצעות הפונקציה ()fread, ואז מודפסים חברי המבנה באמצעות הוראות ()fprintf. הוראת ()fread יכולה לקרוא מחרוזות שכוללות רווחים, ולכן היא עדיפה על ()fscanf.

ההוראה

```
fprintf(ptr,"\n\n");
```

מחדירה שתי שורות ריקות נוספות בין כל אחד מתקליטי הקומפקט-דיסק.

עיצוב התוכנית

השליטה בכתיבת קבצים ובקריאתם מאפשרת לך ליצור יישומים עסקיים ואישיים מתוחכמים. תוכניות ההדגמה בפרק זה שהזינו נתונים מתוך קובץ, קראו את הקובץ כולו. לעומת זאת, לעתים יש לטפל בנתונים בדרכים אחרות.

לדוגמה, ייתכן שעליך לאתר רשומה מסוימת בקובץ. במקרה כזה, יש לפתוח את הקובץ במצב r ולעשות שימוש בלולאה כדי להזין את הנתונים, או כמבנה-אחר-מבנה, או שורה-אחר-שורה, בהתאם לאופן בו נוצר הקובץ. בכל מעבר בלולאה אנו בוחנים את הערך לעומת הנתונים אותם אנו מחפשים. כדי לבחון ערכי מחרוזת, השתמש בפונקציה strcmp()‎, אם היא כלולה בקומפיילר שברשותך. ברגע שמתגלית התאמה, הצג את הנתונים וסגור את הקובץ.

פונקציות הקובץ שמתוארות כאן מבצעות פעולות *סדרתיות*, כלומר - הן קוראות את הקובץ מתחילתו, על-פי סידרו. ניתן להשתמש במצב a כדי להוסיף נתונים לקצהו של הקובץ, אולם אינך יכול לגשת ישירות אל מיקום מסוים בקובץ ולשנות את הנתונים המאוחסנים בו.

אין פירוש הדבר שאינך יכול לשנות את הנתונים השמורים בקובץ, אולם לשם כך עליך להשתמש באלגוריתם שמטפל בקובץ באופן סדרתי. נכיר את האלגוריתם הזה ואלגוריתמים נוספים, בדוגמאות התכנות המפורטות בפרק הבא.

פונקציות גישה-אקראית

קומפיילרים של C++‎ וכן קומפיילרים רבים של ANSI C כוללים פונקציות המשמשות לגישה אקראית לקבצים. *גישה אקראית* פירושה שיש באפשרותך לגשת ישירות אל מיקום מסוים בקובץ ולקרוא את הנתונים המאוחסנים שם, או לשנותם. הפונקציה fseek()‎, לדוגמה, מביאה את המצביע אל מיקום מסוים בקובץ. הפונקציה fread()‎ הבאה אחריה נוטלת את הנתונים מאותו מיקום. הפונקציה ftell()‎ מדווחת על מיקומו הנוכחי של המצביע, והפונקציה rewind()‎ מחזירה את המצביע אל תחילת הקובץ.

שאלות

1. מהו חוצץ קובץ?
2. כיצד משתמשים במבנה קובץ?
3. מדוע מכריזים על קובץ כעל מצביע?
4. הסבר את ההבדלים בין מצבי r, w ו-a.
5. מדוע יש לסגור קובץ לפני סיום התוכנית?
6. כיצד יוצרים פלט של נתונים מספריים?
7. כיצד מדפיסים נתונים במדפסת?
8. מהו ההבדל בין הפונקציה ()fprintf לפונקציה ()fwrite?
9. כיצד מדפיסים מבנה?
10. מהי מטרתה של הפונקציה ()sizeof?

תרגילים

1. כתוב תוכנית שמשתמשת בפונקציה ()fputs ליצירת קובץ של שמות סרטים.
2. כתוב תוכנית שקוראת את שמות הסרטים (מתרגיל 1) לתוך מערך מחרוזת.
3. כתוב תוכנית שעושה שימוש בפונקציה ()fprintf ליצירת קובץ מלאי מוצרים שמכיל את שמו, מחירו וכמותו של כל מוצר.
4. כתוב תוכנית שקוראת את קובץ מלאי המוצרים שיצרת בתרגיל 3.
5. כתוב את התוכניות מתרגילים 3 ו-4 כך שיקראו ויכתבו את הנתונים כמבנה.
6. הסבר מה לקוי בתוכנית הבאה:

```
#include "stdio.c"
main()
{
    FILE fp;
    char letter;
    if((fp = fopen("MYFILE","w"))==NULL)
    {
```

```
        puts("Cannot open the file");
        exit();
    }
    do
    {
        letter=getchar();
        fputc(letter, fp);
    }
    while(letter!='\n');
    fclose(fp);
}
```

13

קשירת כל הקצוות

עתה, לאחר שהכרת את המרכיבים הבסיסיים של C ושל ++C, באפשרותך לכתוב תוכניות. הטכניקות שלמדת בספר זה משמשות אבני הבניין. כדי לפתח תוכנית, יש ליישם את הטכניקות האלה על-פי הסדר הנכון, ולבנות את התוכנית כסדרה של הוראות, גושים של הוראות, ופונקציות.

בפרק אחרון זה נעסוק בפיתוחו של יישום שלם. תגלה שהמוצר הסופי כולל את הפונקציות והפקודות שכבר למדת.

היישום

בפרקים הקודמים עבדנו עם מבנה ועם קובץ כדי לקטלג אוסף של תקליטי קומפקט-דיסק. למדת כיצד ליצור קובץ וכיצד לקרוא קובץ שמכיל נתונים על אוסף התקליטים. עם זאת, תוכניות ההדגמה בפרקים אלה כוללות רק את רכיביו החיוניים של היישום. לא היתה אפשרות למחוק תקליט מהאוסף, או לבצע חיפוש באוסף אחר הנתונים של תקליט מסוים.

התוכניות שניצור בפרק זה משלימות את היישום, ומאפשרות לו לבצע את שבע המטלות הבאות:

- להוסיף תקליט קומפקט-דיסק לאוסף.
- למחוק תקליט מהאוסף.
- לשנות את שמו של התקליט או פרטים אחרים.
- לשנות את מספר החריץ בו מאוחסן התקליט על-גבי המדף.
- לאתר נתונים של תקליט מסוים.
- למיין את סדר התקליטים בקובץ כדי שיתאים לסדר בו הם מאוחסנים על-גבי המדף.
- להדפיס רשימה של התקליטים הכלולים באוסף.

נבנה את היישום כסדרה של פונקציות. אולם לפני קריאת כל אחד מהסעיפים בפרק, התבונן בכותרת הסעיף ורשום לעצמך את ההוראות שתזדקק להן לצורך פיתוח היישום. לדוגמה, התחל בכך שתרשום לעצמך את ההכרזות הכלליות להן תזדקק - קבועים, מצביעים או מבנים. לאחר מכן, קרא את הסעיף כדי לראות כיצד הוא מפתח את התוכנית. זכור, ישנן דרכים רבות לכתוב את התוכנית. ההוראות שלך עשויות להיות שונות מההוראות שנתנו בספר, אך אין להסיק מכך שהשיטה בה נקטת היא שגויה.

בתקליטון המצורף לספר תמצא רישום של היישום המלא. אם התקנת את קובצי ההדגמה בעת שהתקנת את קומפיילר ה-PCC, התוכנית תימצא במחיצה FIRSTC\SAMPLES, תחת השם CDAPP.C. (למידע נוסף על התקנת קומפיילר מ-PCC וגישה לתוכניות ההדגמה - עיין בנספח). עליך להדפיס את רישום התוכנית לפני שתפנה ללימוד הפרק. כך תוכל לעיין בתוכנית אחרי קריאת כל סעיף ולראות כיצד האלגוריתמים מיושמים.

רשומות ומבנים

בפרק זה תיתקל במונח "רשומה" פעמים רבות. "רשומה" היא מונח המשמש בניהול מסדי נתונים, ותוכנית זו היא למעשה יישום בסיסי של ניהול מסדי נתונים. מסד נתונים הוא אוסף של מידע אודות קבוצה של פריטים. במקרה שלנו, הקבוצה היא אוסף תקליטי הקומפקט-דיסק, אולם מסד נתונים יכול לשמש לצורך מעקב אחר לקוחות עסקיים, לניהול מלאי של חברה, או לצורך תיעוד אוספים מכל הסוגים.

דמיין לעצמך את מסד הנתונים כגרסה אלקטרונית של כרטסת נתונים. כל כרטיס נקרא *רשומה*, ומכיל את כל הנתונים על פריט מסוים. לפיכך, כל רשומה בתוכנית שעוסקת בתקליטי קומפקט-דיסק כוללת את המידע אודות תקליט מסוים.

כאשר תוכנית עושה שימוש במבנה, כל רשומה היא *מקרה* אחד של המבנה. בכל פעם שהתוכנית קוראת מבנה מהכונן, היא קוראת רשומה אחת. באופן דומה, בכל פעם שהתוכנית כותבת מבנה לדיסק, היא כותבת רשומה אחת.

הכרזות כלליות

המטלה הראשונה היא להכריז על המשתנים הכלליים הנחוצים. מכיוון שנשתמש בקבצים, יש לכלול את קובץ הכותרת stdio.h:

```
#include "stdio.h"
```

כפי שתראה בהמשך, נשתמש בשלושה קבצים: קובץ הדפסה ושני קובצי כונן. קובץ כונן אחד מכיל את הנתונים אודות אוסף התקליטים, ואילו הקובץ השני משמש כמקום אחסון זמני. אנו מגדירים את שמות הקבצים כקבועים, כדי שנוכל לאזכר אותם על ידי שימוש בשמות קבועים:

```
#define FILENAME "CDfile"
#define TEMPFILE "Temp"
```

תוכל להשתמש בכל שם שתרצה עבור קובצי הכונן שלך, ושמות הקבצים יכולים לכלול גם את הנתיב המלא, כך:

```
#define FILENAME "C:\DATA\CD.DAT"
```

בנוסף, היישום שלנו מניח שעל-גבי המדף יש מקום אחסון ל-20 תקליטים, ולכן נגדיר קבוע שישמש למניעת הזנה של מספרים גדולים מ-20:

```
#define MAX 20
```

כמובן שבאפשרותך להגדיל את המספר בהתאם לצרכיך.

כל שלושת הקבצים מוגדרים בביטוי אחד:

```
FILE *fp, *tp, *printer;
```

כעת, יש לפנות להכרזת המבנה CD, תוך שימוש בהגדרה שהודגמה בפרק 11:

```
struct CD
 {
 char   name[20];
 char   description[40];
 char   category[12];
 float  cost;
 int    number;
 } disc;
```

לבסוף, תזדקק למנגנון כלשהו שימנע מהמשתמש להקצות יותר מתקליט אחד לכל עמדת אחסון במדף. בתוכנית זו נשתמש במערך של מספרים שלמים כדי להקצות את מספרי העמדות, ובמשתנה מטיפוס מספר שלם כדי לציין את המספר הכללי של תקליטי קומפקט-דיסק באוסף:

```
int slots[MAX];
int count;
```

הפונקציה ()main

כל אחת מהמטלות העיקריות של התוכנית תבוצע בפונקציה משלה. בפונקציה ()main אנו זקוקים לדרך כלשהי לבחירת המטלה שתבוצע, ולקריאה לפונקציה המתאימה. הדרך היעילה ביותר לבחירת המטלה היא להציג תפריט של המטלות. כך, ()main צריכה לבצע את הפעולות הבאות:

- לחזור ולהציג את התפריט, עד שהמשתמש יבחר להפסיק את התוכנית.
- להציג את רשימת המטלות.
- לקבל את ההקשה במקלדת לבחירת מטלה.
- לקרוא לפונקציה לביצוע המטלה.
- לעצור את התוכנית כשהמשתמש בוחר באפשרות זו.

כדי לייצג את הבחירה בתפריט, אנו זקוקים למשתנה שיהיה מקומי ל-()main. כאשר המשתמש בוחר בפריט כלשהו מהתפריט (על ידי הקשת מספר), ()main תקרא את הערך

הזה לתוך משתנה ותשתמש בו כדי לקבוע לאיזו פונקציה עליה לקרוא. לפיכך, אנו מתחילים את הפונקציה ()main בהכרזה הבאה:

```
char select;
```

עתה, קודם להצגת התפריט, עלינו לדעת אילו עמדות במדף התקליטים כבר הוקצו. נעשה זאת על ידי קריאה לפונקציה ()getslots. הפונקציה פותחת את הקובץ, קוראת את כל הרשומות, מקצה כל אחד מהערכים של disc.number לרכיב במערך []slots, ומונה את המספר הכולל של רשומות בקובץ:

```
getslots()
{
    int index;
    index=0;
    count=0;
    if((fp = fopen(FILENAME,"r")) != NULL)
    {

    while(fread(&disc, sizeof(disc), 1, fp)==1)
    {
        slots[index]=disc.number;
        index++;
        count++;
    }
    fclose(fp);
    }
}
```

נשאלת כעת השאלה מדוע יש לחזור ולהציג את התפריט? אם לא נחזור על התפריט בתוך לולאה, התוכנית תסתיים אחרי ביצוע פעולה אחת. אם המשתמש רוצה לבצע שתי פעולות, כמו להוסיף תקליטים חדשים ולהדפיס רשימה מעודכנת, יהיה עליו להתחיל את התוכנית פעמיים. מכיוון שאיננו יודעים מראש כמה פעמים יש לחזור על התפריט, אנו משתמשים בלולאה do.

התפריט יכלול שמונה אפשרויות - אחת לכל אחת מהמטלות העיקריות של התוכנית, ואפשרות שמינית לסיום התוכנית. השתמש בהוראות ()puts או ()printf כדי להציג את התפריט, ואז בפונקציה ()getchar כדי להזין את תגובת המשתמש. השימוש ב-()getchar

מונע את הצורך להקיש Enter אחרי בחירת האפשרות הרצויה. אם נשתמש במספרים לצורך הקלט, במקום באותיות, נהיה פטורים מהצורך לטפל באותיות קטנות וגדולות. לפיכך, נוכל להתחיל את ()main כך:

```
main()
{
    char select;
    do
    {
    puts("       My CD Collections Program\n");
    puts("1  Add CD\n");
    puts("2  Delete CD\n");
    puts("3  Edit info CD\n");
    puts("4  Change location\n");
    puts("5  Sort CDs\n");
    puts("6  Locate CD\n");
    puts("7  Print List\n");
    puts("8  Exit Program\n");
    printf("Please enter selections: ");
    select = getchar();
    putchar('\n');
```

מרגע שהמשתמש בוחר בפריט תפריט כלשהו, ()main חייבת לקרוא לפונקציה המתאימה. עבור שמונה אפשרויות תפריט אנו זקוקים לשבעה ביטויי if. כדי להשיג שליטה רבה יותר, כמו גם תוכנית קלה יותר לקריאה, השתמש בפקודת switch שבוחנת את הערך של משתנה הקלט. אל תשכח לכלול קטע של ברירת מחדל, למקרה שהמשתמש מקליד מספר או תו שאינם מופיעים בתפריט. נשלים את ()main כך:

```
switch(select)
{
    case '1' :
        addcd();
        break;
    case '2':
        delcd();
```

```
            break;
        case '3':
            chcd();
            break;
        case '4':
            chloc();
            break;
        case '5':
            sort();
            break;
        case '6':
            locate();
            break;
        case '7':
            plist();
            break;
        case '8':
            break;
        default:
            puts("Invalid entry, try again\n\n");
    }
    }
        while(select!='8');
        return(0);
    }
```

לולאת ה-do...while חוזרת על התפריט (כלומר, ביטוי ה-switch) כל עוד המשתמש לא מקיש 8 לסיום התוכנית. אחרי בחירה באחת מאפשרויות התפריט והשלמת ביצוע המטלה, התפריט שב ומופיע על-גבי המסך, כדי שניתן יהיה לבחור אפשרות נוספת.

הוספת רשומה: הפונקציה addcd()

הפונקציה המשמשת להוספת נתונים אודות תקליט בנויה במתכונת דומה לתוכנית להוספת נתונים שראינו בפרק 12.

עם זאת, ישנם שלושה הבדלים בין הפונקציה addcd() המשמשת ביישום הזה, לבין התוכנית המופיעה בפרק 12. ביישום הזה, המשתמש אינו יכול להוסיף תקליט חדש לאוסף אם אין עמדת אחסון פנויה על-גבי המדף. הפונקציה getslots() מחשבת את מספר התקליטים שכבר אוחסנו. כאשר מגיעים למספר המירבי, מופיעה הודעה על-גבי המסך, והתוכנית חוזרת לתפריט:

```
int pause;
if(count>=MAX)
 {
    puts("Sorry. Your cabinet is full\n");
    pause=getchar();
    return;
 }
```

כמו כן, ביישום זה הקובץ נפתח במצב a, כדי שייווצר - במידה שהוא עדיין אינו קיים.

```
if((fp = fopen(FILENAME,"a")) == NULL)
    {
          printf("Can't open the file %s\n",FILENAME);
          exit();
     }
```

אם הקובץ כבר קיים, נתונים חדשים יתווספו אל קצהו.

פונקציה זו דורשת גם קלט של מספר עמדה, בין 1 ל-20. מכיוון שקלט זה דרוש לכמה ממטלות התוכנית, אנו יכולים לבצעו בפונקציה נפרדת, שנקראת getslot(), ולקרוא לה בשעת הצורך:

```
getslot()
{
int index, flag, pause;
    do
    {
    flag=0;
```

```
printf("Enter the slot number:");
scanf("%d", &disc.number);
for(index=0;index<count;index++)
    {
      if(slots[index]==disc.number)
      {
        printf("Sorry that slot is used. Try again\n");
        flag=1;
      }
      }
  }
  while(disc.number < 1 || disc.number>MAX || flag==1);
  count++;
  slots[count]=disc.number;
return;
}
```

הפונקציה ()getslot מונעת מהמשתמש להקליד מספר עמדה שכבר הוקצתה.

אחרי שהתקליט החדש נכתב לקובץ, נקראת הפונקציה ()getslot כדי לעדכן את מערך העמדות.

מחיקת רשומה: הפונקציה ()delcd

הפונקציה המשמשת למחיקת נתונים אודות תקליט קומפקט-דיסק באוסף, עושה שימוש באלגוריתם תקני לעבודה עם קבצים סדרתיים. בגישה סדרתית, אין אפשרות לגשת ישירות למיקום כלשהו בקובץ כדי לשנות רשומה. כאשר פותחים קובץ במצב w, תוכנו נמחק. אם נפתח את הקובץ במצב a, נוכל רק להוסיף נתונים לקצהו.

הפתרון הוא להשתמש בשני קבצים. הפונקציה ()delcd פותחת את קובץ הנתונים במצב r, ופותחת גם קובץ זמני נוסף במצב w.

```
if((fp = fopen(filename,"r")) == NULL)
{
    printf("Can't open the file %s\n",filename);
    exit();
```

```
}
if((tp = fopen(tempfile,"w")) == NULL)
{
      printf("Can't open the file %s\n",tempfile);
      fclose(fp);
      exit();
}
```

לאחר מכן אנו מזינים את שמו של התקליט שברצוננו למחוק:

```
printf("Please enter the name of the CD: ");
gets(delname);
```

לולאת while קוראת כל מבנה (רשומה) בקובץ המקורי:

```
while(fread(&disc, sizeof(disc), 1, fp)==1)
      {
```

אם אין זו הרשומה שברצונך למחוק, ()delcd כותבת אותה לקובץ הזמני:

```
if(strcmp(disc.name, delname) != 0)
fwrite(&disc, sizeof(disc), 1, tp);
```

יישום זה עושה שימוש בפונקציה ()strcmp במספר מקומות, כדי לקבוע אם נקראה הרשומה הנכונה מהקובץ. הפונקציה ()strcmp אינה כלולה אומנם ב-K&R C, אולם ניתן למצוא אותה בכל הספריות של ANSI C ושל ++C.

כאשר נמצאת הרשומה הנכונה, ()delcd אינה כותבת אותה לקובץ הזמני ובמקום זאת מציבה דגלון, כדי לדווח בהמשך אם נמחק מבנה:

```
else
fflag='y';
}
```

אנו מבצעים את התהליך הזה בלולאת while כדי להמשיך ולקרוא מקובץ אחד ולכתוב לקובץ אחר, עד לסופו של הקובץ. בנקודה זו אנו סוגרים את שני הקבצים:

```
fclose(fp);
fclose(tp);
```

אם ()delcd לא מצאה את הרשומה שרצית למחוק, היא מציגה הודעה על-גבי המסך, ומסתיימת:

```
puts("\nDid not find the CD");
pause=getchar();
```

כעת יש ברשותך שני קבצים שמכילים אותם נתונים. החיסרון היחיד ברוטינה הזו הוא שחייב להיות די מקום בכונן לאחסון שני קובצי נתונים מלאים. ניתן לראות בקובץ הזמני מעין קובץ גיבוי. אם דבר-מה משתבש בקובץ המקורי, הקובץ הזמני יכיל את הנתונים, מעודכנים עד הפעולה הסדרתית האחרונה. אם אינך מעוניין לשמור את הקובץ הזמני על-גבי הכונן, תוכל לפתוח אותו במצב כתיבה ולסגור אותו מייד, כך:

```
tp = fopen(tempfile,"w");
fclose(tp);
```

פעולה זו מותירה את הקובץ בכונן, אך כשהוא ריק לחלוטין, ולפיכך - תופס מעט מאד מקום.

אם אכן מחקת רשומה, יש ברשותך עתה שני קבצים שונים. הקובץ המקורי מכיל את כל הנתונים, כולל הרשומה שהיה ברצונך למחוק. מכיוון שהפונקציה delcd() לא כתבה את המבנה שרצינו למחוק, הקובץ הזמני כולל רק את הנתונים שהתכוונו לשמור. אולם אי אפשר להשאיר את הנתונים במצב כזה. התוכנית ערוכה להוספת נתונים ולביצוע פעולות נוספות תוך שימוש בקובץ המקורי. לפיכך, יש להחזיר את הנתונים המעודכנים לקובץ הנתונים.

קומפיילרים מסוימים של C++ כוללים פונקציה מובנית לשינוי שמו של קובץ, אך לא כולם. כדי לנקוט בגישה כללית, ניתן לפתוח מחדש את שני הקבצים במצבים הפוכים. כלומר, אנו פותחים את הקובץ המקורי במצב w, מוחקים את הנתונים המאוחסנים בו, ופותחים את הקובץ הזמני במצב r. הפונקציה openwr(), שנקראת על ידי delcd() ופונקציות אחרות, מבצעת את הפעולה הזו:

```
if((fp = fopen(filename,"w")) == NULL)
    {
    printf("Can't open the file %s\n",filename);
    exit();
    }
if((tp = fopen(tempfile,"r")) == NULL)
    {
    printf("Can't open the file %s\n",tempfile);
    fclose(fp);
    exit();
    }
```

לאחר מכן, הפונקציה delcd() קוראת את הרשומות מהקובץ הזמני וכותבת אותן לקובץ המקורי. כשהפונקציה סוגרת את הקבצים, הקובץ המקורי מכיל את הנתונים המעודכנים, ללא הרשומות שנמחקו:

```
while(fread(&disc, sizeof(disc), 1, tp)==1)
        fwrite(&disc,sizeof(disc), 1, fp);
fclose(fp);
fclose(tp);
```

לפני שהפונקציה ()delcd מסתיימת, התוכנית קוראת לפונקציה ()getslots, כדי לעדכן את מערך העמדות המוקצות.

היישום ממשיך לחפש רשומה מתאימה גם אחרי מחיקת רשומה. כך ניתן למחוק בעת ובעונה אחת מספר רשומות בעלות שם משותף - אם החלטת למחוק מהאוסף את כל תקליטי הדיסקו, לדוגמה.

ניתן להרחיב את הרוטינה על ידי הצגת הרשומה והנחיה למשתמש לאשר את המחיקה. אם המשתמש מחליט שאין ברצונו למחוק את הרשומה, הרוטינה המורחבת תכתוב אותה לקובץ.

פונקציות שימושיות

חלק מהרוטינות שמוסברות כאן מבוצעות על ידי כמה מהפונקציות העיקריות בתוכנית. במקום לחזור ולכתוב אותן מחדש בכל פעם, ניתן להציב את הרוטינות בפונקציות נפרדות שייקראו בשעת הצורך. אלו הן הפונקציות הבאות:

- **()openwr: פותחת את קובץ הנתונים לקריאה ואת הקובץ הזמני לכתיבה.**
- **()openrw: פותחת את קובץ הנתונים לכתיבה ואת הקובץ הזמני לקריאה.**
- **()nofind: מציגה הודעה כאשר הרשומה המבוקשת אינה קיימת.**

שינוי נתוני תקליט: הפונקציה ()chcd

כדי לשנות את הנתונים של אחד התקליטים באוסף, אנו זקוקים לאלגוריתם הבסיסי ששימש אותנו למחיקת נתונים: יש לכתוב את כל הנתונים לקובץ זמני, ואז לקרוא את הנתונים שלא שונו בחזרה אל קובץ הנתונים. עם זאת, במקום לדלג על הרשומה שברצונך לשנות, יש להזין את הנתונים החדשים עבור רשומה זו, ואז לכתוב אותה לקובץ הזמני.

הפונקציה ()chcd מבקשת תחילה מהמשתמש את שמו של התקליט שברצונו לשנות, ואז מתחילה לולאת while לקרוא כל אחד מהמבנים:

```
openrw();
puts("Change Disc Function\n");
        printf("Please enter the name of the CD: ");
        gets(chname);

        while(fread(&disc, sizeof(disc), 1, fp)==1)
        {
```

לאחר מכן, אם אין זה המבנה שברצונך לשנות, הפונקציה כותבת אותו לקובץ הזמני:

```
if(strcmp(disc.name,chname)!=0)
fwrite(&disc, sizeof(disc), 1, tp);
```

כשמאותרת הרשומה הרלוונטית, הפונקציה chcd() מציגה את הנתונים הנוכחיים, ומבקשת מהמשתמש קלט של הנתונים החדשים:

```
else
{
fflag='y';
puts("Old data\n");
showdisc();
puts("New Data\n");
printf("Please enter the name of the CD: ");
gets(disc.name);
printf("Enter the description:");
gets(disc.description);
printf("Enter the category:");
gets(disc.category);
printf("Enter the cost:");
scanf("%f", &disc.cost);
if(count>=MAX)
   {
    puts("Sorry. You cannot change the location\n");
    pause=getchar();
   }
    else
   {
    getslot();
   }
```

ההוראה if(count>=MAX) מונעת מהמשתמש להקליד מספר עמדה חדש במקרה שכל עמדות האחסון מלאות. כשהמדף מלא, הפונקציה ()getslot אינה מתבצעת, ומספר העמדה המקורי שנקרא מהכונן נכתב עם יתרת הרשומה הערוכה.

השימוש בפונקציה ()chcd מחייב את המשתמש להקליד מחדש את כל הנתונים אודות התקליט, אפילו אם ברצונו לשנות פריט אחד בלבד. ניתן להתאים את היישום כך שהמשתמש יוכל להקיש Enter בלבד כדי לאשר את הערכים הנוכחיים של פריטי הנתונים השונים.

לדוגמה, כדי לאשר את השם החדש של תקליט קומפקט-דיסק, השתמש בהוראות הבאות:

```
printf("Please enter the name of the CD: ");
gets(name);
if(strlen(name)>0)
   strcpy(disc.name,name);
```

אם המשתמש מקליד שם חדש, השם מוקצה ל-disc.name ונכתב לקובץ עם יתרת הרשומה. אם המשתמש מקיש Enter מבלי להקליד שם, תוכנו של disc.name אינו משתנה, והשם המקורי נכתב עם הרשומה.

מכיוון שיהיה עלינו להציג מבנה מספר פעמים בתוכנית, יצרנו את הרוטינה בתוך פונקציה ששמה ()showdisc, והיא נקראת בשעת הצורך:

```
showdisc()
{
     printf("CD Name          %s\n",disc.name);
     printf("Description:     %s\n",disc.description);
     printf("Category:        %s\n",disc.category);
     printf("Cost:            %6.2f\n",disc.cost);
     printf("Location:        %d\n",disc.number);
     puts("\n\n");
     return;
}
```

לאחר קריאת הנתונים החדשים, ()chcd כותבת את המבנה לקובץ הזמני:

```
fwrite(&disc, sizeof(disc), 1, tp);
```

כשסיימה לקרוא את הקובץ fp כולו, ()chcd סוגרת את הקובץ ואת הקובץ tp. אם לא שונתה אף רשומה (כלומר, התקליט המבוקש לא אותר), הפונקציה מציגה הודעה על כך ומסתיימת:

```
fclose(fp);
fclose(tp);
if(fflag=='n')
nofind();
```

לעומת זאת, אם אכן בוצע שינוי ברשומה כלשהי, chcd() פותחת מחדש את הקבצים במצבים הפוכים, ואז כותבת את הנתונים בחזרה לקובץ המקורי ומעדכנת את מערך העמדות:

```
else
    {
    openwr();
   while(fread(&disc, sizeof(disc), 1, tp)==1)
    fwrite(&disc, sizeof(disc), 1, fp);
    fclose(fp);
    fclose(tp);
    }
    getslots()
return;
}
```

שינוי מיקום: הפונקציה chloc()

הפונקציה המשמשת לשינוי עמדה של תקליט באוסף, דומה בעיקרה לפונקציה המשמשת לעריכת הרשומה כולה, אלא שהקלט כולל אך ורק מספר עמדה חדש:

```
puts("Old Data\n");
showdisc();
puts("\nNew Location\n");
getslot();
```

עם זאת, הוראת if בתחילתה של הפונקציה, מונעת מהמשתמש לשנות את מיקום התקליט במקרה שמדף האחסון כבר מלא.

הצגת רשומה: הפונקציה ()locate

כדי להציג רשומה, אנו מזינים את שמו של התקליט הרצוי, פותחים את הקובץ לקריאה, וקוראים כל אחת מהרשומות עד לאיתור הרשומה הרצויה:

```
while(fread(&disc, sizeof(disc), 1, fp)==1)
    {
    if(strcmp(disc.name,name)==0)
    {
```

שילוב הפונקציות

היישום בספר זה מציג כל אחת מהמטלות כפונקציה נפרדת. עם זאת, מכיוון שהפונקציות כמעט זהות זו לזו, ניתן לשלבן לפונקציה אחת. אחרי איתור הרשומה והצגתה, גרסה מתוקנת של התוכנית תעשה שימוש בפקודת if כדי לקבוע איזה חלקים מהמבנה יש להזין:

```
if(select = '3')
{
        puts("New Data\n");
        printf("Please enter the name of the
        CD: ");
        gets(disc.name);
        printf("Enter the description:");
        gets(disc.description);
        printf("Enter the category:");
        gets(disc.category);
        printf("Enter the cost:");
        scanf("%f", &disc.cost);
}
getslot();
```

אם בחרת 3 כדי לערוך את נתוני התקליטים, יופיעו שורות הנחיה להזין את השם, התיאור, הקטגוריה והמחיר. אחרי פקודת ה-if, התוכנית קוראת לפונקציה ()getslot כדי להזין מספר עמדה חדש. אם בחרת 4, כדי לשנות את המיקום בלבד, יכלול הקלט אך ורק עמדה חדשה.

()locate קוראת לפונקציה ()showdisc כדי להציג את הנתונים, ואז משהה את התצוגה כדי לאפשר למשתמש לקרוא את תוכן המסך:

```
        fflag='y';
        showdisc();
        printf("Press Enter to continue");
        pause=getchar();
        putchar('\n');
        }
}
```

הפונקציה ()showdisc קוראת את הקובץ כולו, ולפיכך יוצגו כל התקליטים הטשאים אותו שם.

הדפסת דו"ח: הפונקציה ()plist

כדי להדפיס רשימה של אוסף התקליטים, על התוכנית לפתוח שני קבצים: קובץ הנתונים לצורך קריאה, וקובץ מדפסת - באמצעות "prn", שם הקובץ התקני של DOS - לכתיבה:

```
if((fp = fopen(filename,"r")) == NULL)
{
    printf("Can't open the file %s\n",filename);
    exit();
}
if((printer = fopen("prn","w")) == NULL)
{
    printf("Something is wrong with the printer\n");
    fclose(fp);
    exit();
}
```

הפונקציה ()plist קוראת כל אחת מהרשומות ומדפיסה את הנתונים בעזרת פונקציות ()fprintf:

```
while(fread(&disc, sizeof(disc), 1, fp)==1)
{
fprintf(printer,"CD Name        %s\n",disc.name);
fprintf(printer,"Description:   %s\n",disc.description);
fprintf(printer,"Category:      %s\n",disc.category);
fprintf(printer,"Cost:          %6.2f\n",disc.cost);
fprintf(printer,"Location:      %d\n",disc.number);
fprintf(printer,"\n\n");
}
```

לבסוף, הפונקציה סוגרת את שני הקבצים וחוזרת ל-()main:

```
    fclose(printer);
    fclose(fp);
    return;
}
```

מיון רשומות: הפונקציה ()sort

בשעת הוספת תקליטים לאוסף, אינך מציב אותם תמיד בעמדות סמוכות על-גבי מדף האחסון. ייתכן שברצונך להשאיר עמדות פנויות, שמורות לתקליטים מיוחדים. כלומר, כאשר מוסיפים רשומות לקובץ, ייתכן שסדרן שונה מסדר התקליטים על-גבי המדף.

כאשר מדפיסים דו"ח, התקליטים יופיעו על-פי הסדר בו הזנת אותם לקובץ. כדי לאתר תקליט לפי מספר העמדה, עליך לסרוק את הרשימה כולה.

מיון הרשימה במקרה זה פירושו כתיבתה מחדש, כך שהתקליטים יאוחסנו על-פי סדר מספרי העמדות. התקליט בעמדה מספר 1 - ראשון, אחריו התקליט מעמדה מספר 2, וכן הלאה.

יש עשרות דרכים למיון קובץ. אלגוריתמים אחדים קוראים את כל הנתונים לתוך מערך, ממיינים את מרכיבי המערך בזיכרון, ואז כותבים את המערך בחזרה לכונן. אלגוריתמים אחרים משתמשים בשני קבצים, או יותר, מבצעים קריאה וכתיבה באצוות, ואז מאחדים את הקבצים לקובץ אחד, ממוין על-פי הסדר הרצוי.

האלגוריתם המתואר כאן, בפונקציה ()sort, זקוק לקובץ אחד בלבד, אך עושה שימוש במערך של המבנה. יש להכריז על מערך המבנה ועל מספר משתנים:

```
sort()
{
struct CD temp[MAX];
int index, loop1, loop2, endloop;
loop1=0;
loop2=0;
endloop=0;
index=o;
```

הפונקציה ()sort פותחת את הקובץ לקריאה, קוראת את כל הנתונים לתוך המערך וסוגרת את הקובץ:

```
while(fread(&disc, sizeof(disc), 1, fp)==1)
  {
    temp[index]=disc;
    index++;
  }
fclose(fp);
```

המשתנה index משמש כמספר הסידורי.

()sort פותחת מחדש את הקובץ לכתיבה, ומציבה שתי לולאות for כדי לכתוב את הנתונים בחזרה לקובץ:

```
if((fp = fopen(filename,"w")) == NULL)
  {
    printf("Can't open the file %s\n",filename);
      exit();
  }
for(loop1=1;loop1<MAX+1;loop1++)
    {
        for(loop2=0;loop2<count;loop2++)
```

לולאת ה-for החיצונית מגדילה את loop1 מ-1 ועד המספר המירבי של תקליטים. בחזרה הראשונה על הלולאה החיצונית, סורקת הלולאה הפנימית את המערך כולו כדי לאתר תקליטים בעמדה 1. כל תקליט שהיא מוצאת שמספר העמדה שלו הוא 1 נכתב לקובץ:

```
if(temp[loop2].number==loop1)
    {
    fwrite(&temp[loop2], sizeof(temp[loop2]), 1, fp);
    endloop++;
    }
```

בחזרה הבאה על הלולאה החיצונית סורקת שוב הלולאה הפנימית את המערך כולו, אך הפעם כדי לאתר תקליט בעמדה 2, וכותבת אותו לקובץ. התהליך חוזר על עצמו עבור כל מספרי העמדות האפשריים.

עם זאת, מרגע שכתבנו את כל התקליטים בחזרה לקובץ, אין עוד צורך להמשיך ולחפש מספרי עמדות. אנו משתמשים במשתנה endloop כדי לעקוב אחר מספר המבנים שנכתבו לקובץ. אחרי כתיבת כל אחד מהמבנים, ()sort מגדילה את ערכו של endloop. כאשר endloop שווה למספר תקליטי הקומפקט-דיסק שבאוסף, פירוש הדבר שכל המבנים נכתבו וניתן להפסיק את החזרות:

```
if(endloop==count)
    break;
```

הקובץ נסגר והפונקציה ()sort מסתיימת.

עיין שוב בקוד המלא של התוכנית. נסה להוסיף בעצמך שיפורים שהיית רוצה לראות בתוכנית.

נספח

השימוש בתקליטון ההדגמה

בשלב זה אתה ודאי משתוקק כבר להתחיל לכתוב תוכניות C. התקליטון המצורף לספר מכיל את כל הדרוש לך לשם כך - עורך מלל לכתיבת תוכניות, קומפיילר וקשר, וכן תוכניות הדגמה שיסייעו לך לצאת לדרך. התקליטון מכיל גם את הפתרונות לתרגילים שמופיעים בסופו של כל אחד מפרקי הספר.

התקנת המהדר (קומפיילר)

כדי להתקין את התוכניות ושאר הקבצים על הכונן הקשיח במחשב שברשותך, בצע את השלבים הבאים:

1. הכנס את התקליטון לכונן התקליטונים.

2. התחבר לכונן על ידי הקשת :A או :B (בהתאם לכונן בו אתה משתמש) והקש Enter.

3. הקלד CINSTALL והקש Enter. יש צורך ב-750K פנויים בכונן הקשיח, לכל הפחות. אם אין די מקום פנוי בכונן הקשיח, CINSTALL תציג אזהרה ותסתיים. מחק קבצים מיותרים והפעל את CINSTALL פעם נוספת.

על המסך יופיע סימן הנחיה שמציין את מחיצת ברירת המחדל בה יותקנו התוכניות - C:\FIRSTC.

4. הקש Enter כדי לאשר את מחיצת ברירת המחדל, או הקלד נתיב חדש והקש Enter לאחר מכן.

5. הקומפיילר, הקשר ועורך המלל יותקנו בכונן הקשיח; תוכנית ההתקנה תשאל אם ברצונך להתקין גם את התוכניות.

6. אם ברצונך להתקין את תוכניות ההדגמה, בחר Yes.

7. תופיע הודעה שמדווחת כי כל התוכניות הותקנו בהצלחה. לאחר מכן תוכנית ההתקנה שואלת אם ברצונך ש-CINSTALL תבדוק את הקובץ CONFIG.SYS. הקומפיילר והקשר זקוקים ל-25 קבצים ול-25 חוצצים, לכל הפחות, ב-CONFIG.SYS. אם תבחר Yes, CINSTALL תבדוק את קובץ התצורה CONFIG.SYS, ותגדיל אותו בהתאם לצורך. אם CINSTALL משנה את הקובץ CONFIG.SYS, העותק המקורי של הקובץ נשמר תחת השם CONFIG.OLD, למקרה שדבר-מה ישתבש.

8. תוכנית ההתקנה תציג לך שאלה אם ברצונך לאתחל מחדש את המחשב. האתחול יישם את השינויים שהוכנסו ב-CONFIG.SYS, כך שניתן יהיה להתחיל לכתוב תוכניות ולקמפל אותן. בחר Yes (או בחר No, אם החלטת לנטוש את ההתקנה, מסיבה כלשהי).

9. הקש Enter כדי לסיים את ההתקנה.

אם החלטת שלא להתקין עתה את תוכניות ההדגמה, תוכל להתקין אותן באופן ידני, בשלב מאוחר יותר. העתק את הקובץ SAMPLES.EXE מהתקליטון אל הכונן הקשיח, היכנס למחיצה אליה העתקת את התוכנית, הקלד SAMPLES והקש Enter.

במקרה שתוכנית ההתקנה CINSTALL לא התקינה את התוכנה כיאות, ניתן להתקינה ידנית. העתק את הקבצים הבאים אל הכונן הקשיח:

PCC12C.EXE

EDITOR.EXE

SAMPLES.EXE

היכנס לכונן אליו העתקת את התוכניות והרץ כל אחת מהן.

השימוש בעורך התוכניות CEDIT

ניתן להשתמש בכל עורך מלל או מעבד תמלילים לכתיבת קוד המקור של התוכניות. עם זאת, אם אתה משתמש במעבד תמלילים, יש לשמור את הקובץ בפורמט ASCII. קובץ ASCII אינו מכיל את הקודים והפקודות לקביעת התצורה, שמשמשים את מעבדי התמלילים בעת שמירת מסמכים. הקומפיילר ינסה לקמפל את הקודים האלה והדבר עלול לגרום להצגת סדרה ארוכה של הודעות שגיאה ואזהרות.

התקליטון המצורף לספר מכיל תוכנית שנקראת CEDIT.COM. זהו עורך מלל פשוט שיכול לסייע לך לכתוב ולערוך תוכניות שאורכן אינו עולה על 1000 שורות.

כדי להפעיל את CEDIT, היכנס למחיצה בה התקנת את הקומפיילר, הקלד CEDIT והקש Enter.

CEDIT אינו כולל תכונות לקביעת התצורה של המלל, ואינו עובר באופן אוטומטי לשורה חדשה כשמגיעים לסופה של השורה הנוכחית. עליך להקיש Enter כדי לעבור לשורה חדשה. אך עם זאת, CEDIT כולל את כל התכונות להן אתה זקוק כדי לכתוב תוכניות.

בחלקו העליון של מסך ה-CEDIT מופיעה שורת מצב, שמציינת את מיקומו של הסמן מבחינת השורה והתו, את שמו של הקובץ שאתה עורך, וכן אם אתה במצב החדרה (Ins) או במצב שכתוב (Ovr). במצב שכתוב, תווים חדשים יתפסו את מקומם של התווים הנוכחיים בשורה. במצב החדרה, תווים חדשים דוחקים את התווים הקיימים.

אורכה המירבי של שורה הוא 78 תווים. כשתתקרב לסופה של השורה, יישמע צפצוף אזהרה, וצפצוף נוסף יישמע כשהשורה התמלאה ואינה יכולה להכיל תווים נוספים. אם אתה מחדיר מלל לשורה קיימת, במצב החדרה, CEDIT עובר אוטומטית למצב שכתוב כשהשורה מתמלאת, כדי למנוע מהמשתמש להקיש יותר מ-78 תווים בשורה.

לא ניתן אומנם לבחור קטעי מלל ולהעתיקם ב-CEDIT, אך תוכל לעשות שימוש במקשים הבאים:

Backspace	מוחק תווים משמאל לסמן.
Del	מוחק תווים מימין לסמן.



PgUp	מעלה את הסמן 20 שורות.
PgDn	מוריד את הסמן 20 שורות.
Home	מביא את הסמן לתחילת השורה.
End	מביא את הסמן לקצה השורה.
Ins	מעביר ממצב החדרה למצב שכתוב, ולהיפך.
F1	מחדיר שורה ריקה.
F2	מוחק שורה שלמה.
F3	מציג מידע סיוע.
F7	מציג אפשרויות לשמירת תוכנית, ניקוי המסך ויציאה מ-CEDIT.
F10	מטעין תוכנית.

השתמש במקשי החצים כדי להניע את הסמן על-פני המסמך. לחיצה על מקש החץ המופנה למטה כאשר הסמן נמצא בסוף המסמך - מוסיפה שורות ריקות, בדיוק כמו הקשת Enter.

שמירת תוכנית

לשמירת מסמך, הקש F7. מופיעה שאלה אם ברצונך לשמור את התוכנית. הקש Y. אם יש כבר לקובץ שם, הוא יישמר באופן אוטומטי תחת אותו שם. אם זוהי תוכנית חדשה, תתבקש להעניק שם לקובץ. הקלד שם לתוכנית כולל הסיומת C., והקש Enter. לא ניתן לשמור תוכנית בשם זהה לשמה של תוכנית שכבר שמורה בכונן.

עורך המלל יציג על-גבי המסך את השאלה אם ברצונך לצאת מ-CEDIT. הקש N כדי להישאר בעורך ולהתחיל, או להטעין, תוכנית חדשה. הקש Y אם ברצונך לנטוש את CEDIT.

ניקוי המסך

אם ברצונך לנטוש תוכנית תוך כדי עריכתה ולנקות את המסך, הקש F7 N N.

יציאה מ-CEDIT

כדי לצאת מ-CEDIT, הקש F7. הקש Y אם ברצונך לשמור את התוכנית שעל המסך, או N אם ברצונך לנטוש את התוכנית. לאחר מכן הקש Y כדי לשוב לסימן ההנחיה של DOS.

הטענת תוכנית

כדי להטעין תוכנית ל-CEDIT - הקש F10, הקלד את שם התוכנית כולל הסיומת, והקש Enter.

ניתן להטעין קובץ רק כאשר המסך ריק. לא ניתן להטעין קובץ אם מופיע מלל כלשהו על המסך, או אם הקלדת מלל כלשהו מאז הפעם האחרונה שניקית את המסך, אפילו אם מחקת אותו. כדי לנקות את המסך ולאפשר הטענת קובץ, הקש N N F7.

הדפסת מלל התוכניות

לא ניתן להדפיס את קוד התוכנית מתוך CEDIT. עם זאת, בין התוכניות שהתקנת בכונן הקשיח כלולה גם תוכנית ששמה CPRINT.EXE, שהותקנה במחיצה בה הותקנו העורך והקומפיילר. כדי להדפיס את קוד התוכנית, עליך להתחבר למחיצה ולהקליד את מלת ההפעלה CPRINT ולאחריה את שם התוכנית, כך:

```
cprint myfirst.c
```

CPRINT תדפיס את התוכנית ותציב את שם הקובץ לפני הקוד.

ניתן להשתמש ב-CPRINT לצורך הדפסת הקוד המלא של התוכנית CDAPP.C שבפרק 13.

שימוש בקומפיילר PCC

קומפיילר PCC שמצורף לספר אינו גרסה חתומה שמשמשת להדגמה, אלא קומפיילר מלא על כל התכונות הדרושות. הוא מכיל ספריית פונקציות נרחבת שכוללת פונקציות לטיפול במחרוזות ופונקציות לגישה-אקראית לקבצים.

בעזרת PCC תוכל ללמוד את שפת C וגם לכתוב יישומים מלאים. PCC כולל גם מאגד (אסמבלר), אם ברצונך ללמוד תכנות בשפת Assembly!

זכויות היוצרים לקומפיילר ולתוכניות הנלוות שייכות לחברת C WARE, אולם הם מופצים כ"שותפה". פירוש הדבר שאתה רשאי לנסות את התוכנית למשך 30 יום, כדי לראות אם היא מתאימה לצרכיך. בסופה של תקופת הבדיקה עליך לרשום את עותק ה-PCC שברשותך, או לחדול מלהשתמש בו.

רישום הקומפיילר, שכרוך בתשלום סך של $30, מעניק לך רשות להשתמש בתוכנית בכל מחשב שברשותך, כל עוד השימוש נעשה במחשב אחד בלבד בעת ובעונה אחת. אתה רשאי גם להעתיק את התוכנית ולמוסרה למשתמשים אחרים לצורך בדיקה. כמשתמש רשום, אתה זכאי לקבל עדכונים כשיוצאות לשוק גרסאות חדשות, בקשר ישיר עם חברת C WARE. בסוף הנספח תמצא טופס שניתן לצלמו ושמשמש לרישום עותק ה-PCC שברשותך.

מוסדות חינוכיים שמעוניינים להשתמש ב-PCC בקורסים ללימוד שפת C, כמו גם מוסדות פרטיים או מסחריים, מתבקשים לפנות לחברת CWARE, או לעיין בקובץ PCC.DOC.

התקנת PCC

עם הרצת תוכנית ההתקנה CINSTALL, כפי שתיארנו בתחילת הנספח, מותקנים הקומפיילר ותוכניות העזר בכונן הקשיח.

אם חסר לך מקום אחסון בכונן הקשיח, תוכל למחוק חלק מהקבצים שהותקנו עם PCC. עם זאת, כדי להריץ את תוכניות ההדגמה הכלולות בספר, תזדקק לקבצים הבאים:

PCC.EXE	מחזור הביצוע הראשון של קומפיילר C.
PC2.EXE	מחזור הביצוע השני של קומפיילר C.
PCCA.EXE	המאגד ומחזור הביצוע השלישי של קומפיילר C.
PCCL.EXE	קשר קובץ העצמים.
PCCS.S	ספריית הפונקציות התקניות של שפת C.
STDIO.H	קובץ כותרת עבור חבילת הקלט/פלט התקנית.

עיין בקובץ PCC.DOC למידע נוסף על קובצי ה-PCC.

ההבדלים לעומת ANSI C

PCC תואם לחלוטין את התוכניות והטכניקות שהודגמו בספר, למעט שני מקרים שוליים:

- PCC אינו מחדיר קוד שורה-חדשה באופן אוטומטי אחרי כל פקודת ()puts. אם שורות הפלט שמתקבל צמודות זו לזו, הוסף n\ לצורך הצגת תווים מילוליים, כך:

```
puts("My CD Collection\n");
```

או החדר את תו השורה-החדשה בפקודות ()putchar, כך:

```
putchar('\n');
```

קובצי ההדגמה בתקליטון המצורף כבר כוללים את קוד השורה-החדשה.

- במקרה של העברת מבנה בין פונקציות, קומפיילר PCC מציג אזהרה. ניתן להתעלם מהאזהרה, והיא תבוטל בגרסאות הבאות של הקומפיילר.

קימפול תוכנית

כדי לקמפל תוכנית, הקלד PCC ואת שמו של קובץ קוד-המקור. לדוגמה, כדי לקמפל תוכנית ששמה VIDEO.C, הקלד:

PCC video

והקש Enter. הקומפיילר מניח שקובצי קוד-המקור הם בעלי סיומת C., וכן שהתוכנית נמצאת במחיצה בה נמצא הקומפיילר. אם שם הקובץ חסר סיומת, הצב נקודה אחרי השם, כך:

PCC NOEXT.

במקרה שסיומת הקובץ שונה מ-C., יש להקליד את הסיומת אחרי השם, כך:

PCC video.pgr

PCC יקמפל את התוכנית וייצור קובץ עצמים בעל סיומת O., ששמו זהה לשמו של קובץ קוד-המקור. הקומפיילר עושה שימוש, למעשה, בשלושה קבצים כדי ליצור את קובץ העצמים: PCC.EXE, PC2.EXE ו-PCCA.EXE. עם זאת, עליך להקליד PCC ואת שם התוכנית בלבד; הקומפיילר עצמו קורא לקבצים הנוספים, בשעת הצורך. אם הקומפיילר מגלה שגיאה, הוא מציג את מספר השורה בה היא התגלתה, את שורת הקוד, והסבר קצר על מהות השגיאה. במקרים מסוימים השגיאה עצמה עשויה להיות בשורה שקודמת לשורה המוצגת. לדוגמה, אם שכחת להציב נקודה-פסיק בסופה של שורה, הלוגיקה של התוכנית תזהה את השגיאה בשורה שבאה אחריה.

בנוסף לשגיאות, קומפיילר PCC מציג גם אזהרות. אזהרות חמורות פחות משגיאות, ובמקרים מסוימים התוכנית תפעל כהלכה למרות האזהרה. עם זאת, יש לבדוק כל שגיאה וכל אזהרה בקפידה. תקן את הבעיה, וקמפל את התוכנית פעם נוספת.

קישור התוכנית

לאחר שהתוכנית קומפלה ללא שגיאות, השתמש בקשר PCCL כדי ליצור תוכנית הרצה בעלת סיומת EXE. הקשר מוסיף את הקוד הדרוש עבור פונקציות מובנות מתוך קובץ הספרייה PCCS.S ומקובצי כותרת אחרים שציינת בתוכנית. PCC כולל שני קובצי כותרת: STDIO.H נחוץ לתוכניות שעושות שימוש בקבצים או במדפסת; MATH.H דרוש לתוכניות שעושות שימוש בפונקציות מתמטיות מורכבות.

כדי לקשר את התוכנית, הקלד PCCL ואת שמו של קובץ העצמים. PCCL מניח שהסיומת של קובץ העצמים היא O. לדוגמה, כדי לקשר את הקובץ video.o (שקומפל מהתוכנית video.c), הקלד:

PCCL video

והקש Enter.

תהליך הקישור יסתיים בהצלחה אם התוכנית קומפלה ללא שגיאות, קובצי הכותרת הדרושים נמצאים על-גבי הכונן, ויש די מקום בכונן לשמירת תוכנית ההרצה.

הרצת התוכנית

אם לא התגלו שגיאות במהלך תהליך הקישור, הרץ את התוכנית על ידי הקלדת שמה והקשת Enter.

שימוש בתוכניות ההדגמה

בזמן התקנת הקומפיילר והעורך, תוכניות ההדגמה שנדונות בספר מותקנות בכונן הקשיח במחיצת המשנה FIRSTC\SAMPLES. כל אחת מהתוכניות מופיעה בשם שהוענק לה בספר. לדוגמה, בפרק 12 תראה תוכנית שמתחילה בשורה:

```
/*fread.c*/
```

פירוש הדבר שתמצא את התוכנית בשם זה במחיצה SAMPLES.

במחיצת תוכניות ההדגמה מופיע גם קוד המקור לתוכנית CPRINT, כדי שתוכל ללמוד אותו בעצמך.

פתרונות לתרגילים

בסופו של כל פרק תמצא תרגילים בכתיבת תוכניות. (ישנן גם שאלות עיוניות, שאת התשובות להן תמצא פשוט על ידי קריאת הקטע המתאים בספר). הפתרונות לתרגילים מאוחסנים במחיצה SAMPLES, לפי הפרק ומספר התרגיל, ומתחילים באות E. לדוגמה, הפתרון לתרגיל מספר 2 בפרק 11 מאוחסן בקובץ E11-2.C.

באפשרותך לעיין בתוכניות או להתאימן לצרכיך על ידי הטענתן לעורך המלל. ניתן גם לקמפל את התוכניות ולהריצן בעזרת הקומפיילר PCC.

אם ברשותך קומפיילר אחר, לעומת זאת, ייתכן שיהיה עליך לערוך תוכניות שמטפלות בקבצים. ייתכן שהקומפיילר שברשותך מחייב הכללה של הקובץ stdio.h בצורת הכתיב הבאה:

```
#include <stdio.h>
```

קרא בקפידה כל הודעת שגיאה שהקומפיילר שברשותך מציג בעת תהליך הקומפילציה. עם זאת, במרבית המקרים תוכל לקמפל ולהריץ את התוכניות ללא עריכה, או עם עריכה מועטה בלבד.

בנוסף לקובצי קוד-המקור שהזכרנו, המחיצה SAMPLES כוללת גם קובץ מלל שמכיל את כל התרגילים ופתרונותיהם, וכן תשובות לשאלות בהן התבקשת לגלות מה לקוי בקוד שהוצג לך. הקובץ נקרא EXERCISE.TXT. ניתן להשתמש בכל מעבד תמלילים כדי לקרוא או להדפיס את תוכנו של הקובץ.

תרשים A.1

Software Registration Form

Remit to:

C Ware Corporation
P.O. Box 428
Paso Robles, CA 94447

PCC version 1.2c

You can also order by phone using your P.O.#, Mastercard or VISA. (805) 239-4620, 9:00 A.M.-4:00 P.M., PST ONLY.

____ PCC Registration (Includes registration software OR latest version of PCC.)

@ $ 30.00 ea $______

____ UPGRADE to the newest version (Includes latest version of the program diskette, with documentation on the disk.)

@ $ 5.00 ea $______

Orders are normally shipped by USPS at no additional charge.

For UPS shipment, please add $3.00

$ 3.00 ea $_____
Subtotal ______
Total $ ______

Payment by: [ ] Check [ ] MC [ ] Visa [ ]

PO # ____________

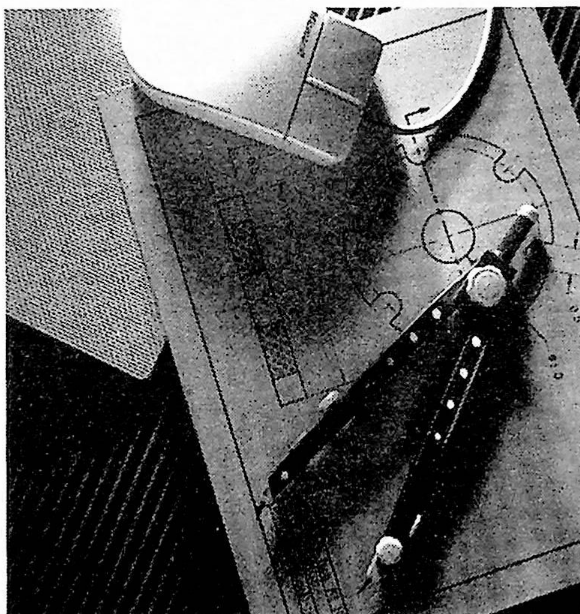
Name: ______________________________
Company: ______________________________
Address: ______________________________
____
____
Day Phone: ______________ Eve: ______________
Card #: ______________ Exp. Date: __________
Signature of cardholder: ______________________

אינדקס

האלף בית של
AutoCAD
גירסה 12

אלן ר. מילר

- הנחיות צעד-אחר-צעד למתחילים
- השלם שרטוט ראשון בזמן שיא
- הוסף לשרטוטים שלך כותרות, שכבות ומימדים



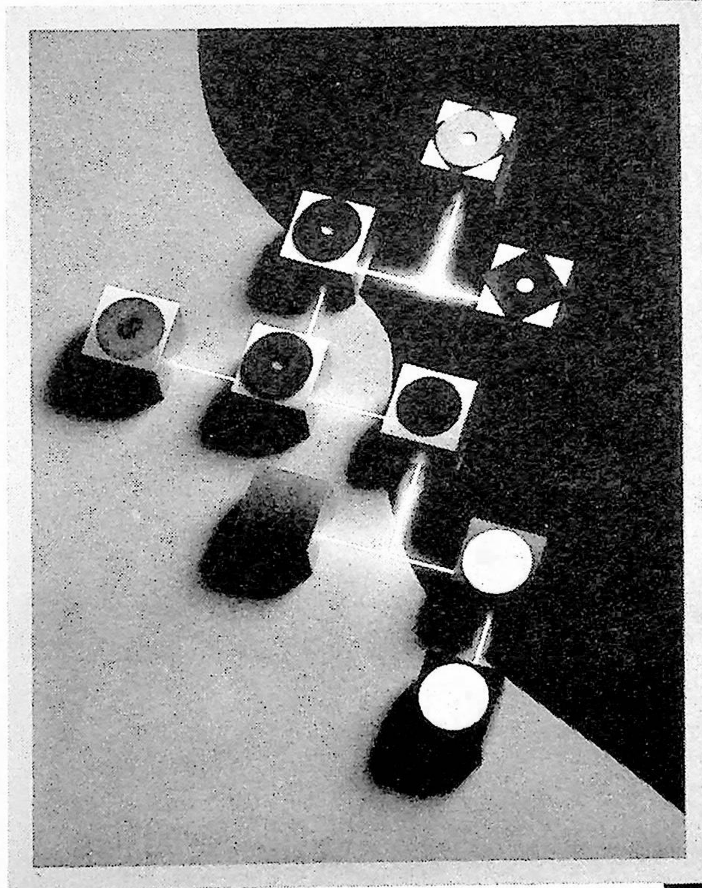
BUG
SYBEX

AUTOCAD 12: זה קל ומשוכלל!

תכנות
ב־Borland Pascal

סקוט ד. פלמר

כולל
תקליטון
התוכנה
חינם



ספר הדרכה מעמיק
עבור כל מתכנתי
DOS ו־Windows

כולל עצות מועילות
וקיצורי דרך
להפיכת התוכניות
לפשוטות וברורות
יותר

בונוס מיוחד!
תקליטון התוכנה
מצורף לספר - חינם

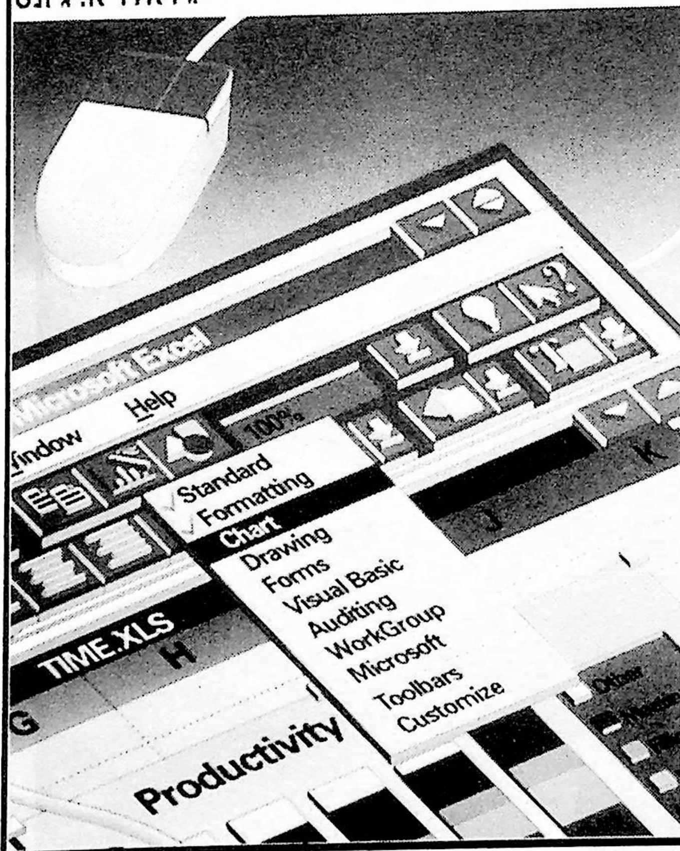


MULTI BUG SYSTEM

Excel
for Windows
Quick & Easy

גרסה 5

ג'ראלד א. ג'ונס



Excel
for Windows
מהיר וקל

▼

מצוין עבור מתחילים

▼

יצירת גיליון עבודה
בתוך דקות

▼

רק הדברים הבסיסיים -
ישר ולעניין

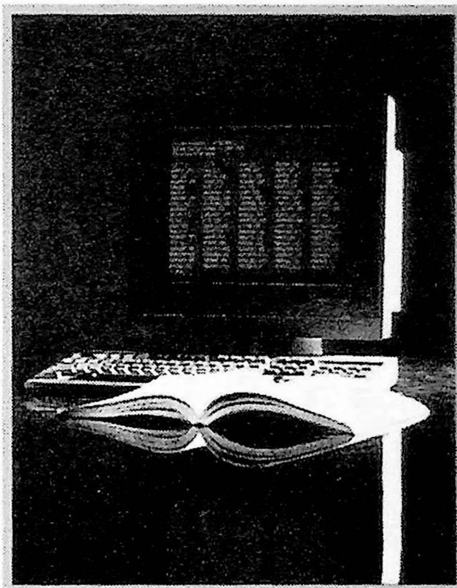
BUG

SYBEX

DOS 6.2

מדריך בזק

מהדורה שנייה



- מדריך מושלם למתחילים ולמומחים כאחד
- הדרכה בשיטת צעד-אחר-צעד
- מאורגן לפי סדר האלף־בית האנגלי להתמצאות נוחה
- מכסה את גרסאות DOS 6.0 ו־6.2

רוברט מ. תומס

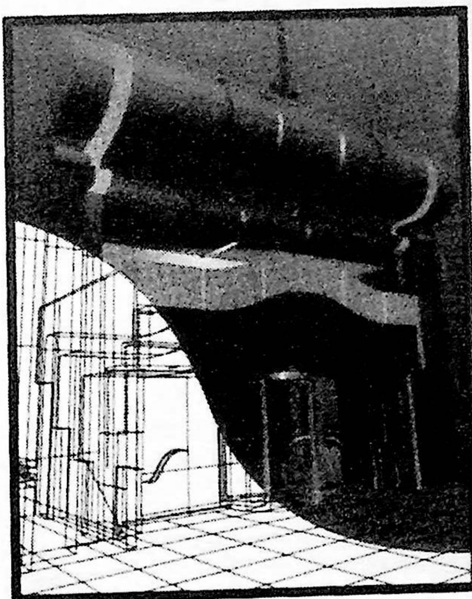
מדריך הכיס המדויק והמקיף ביותר שניתן למצוא





CorelDRAW 4

מדריך בזק



◄ מדריך מהיר וקל לפקודות, פונקציות, ולאמצעים שבהם הינך מרבה להשתמש

◄ כיסוי מלא של התוכניות CorelDRAW, CorelCHART, CorelPHOTO-PAINT, CorelSHOW, CorelTRACE ו־CorelTRACE.

◄ מסודר לפי האלף־בית האנגלי וכתוב בשפה ברורה, תמציתית ולא טכנית.

גורדון פדויק

מדריך הכיס המקיף ביותר כיום עבור CorelDRAW



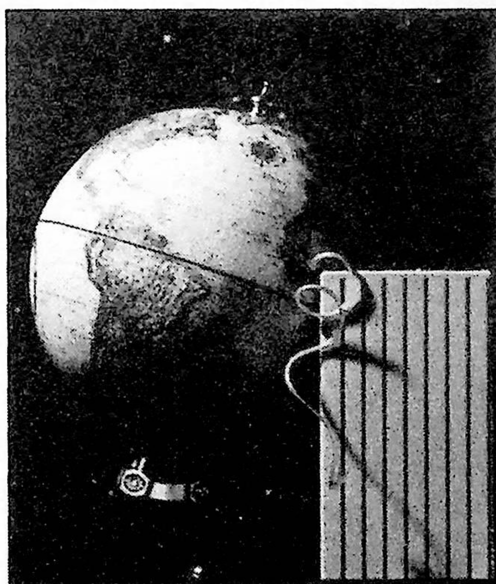


SYBEX

Internet

מדריך בזק

- המדריך המתאים למשתמשים בכל הרמות
- כל ערך מוסבר בשפה פשוטה, בהירה ומדויקת
- מאורגן בסדר האלף בית האנגלי כדי למצוא במהירות את התשובות שאתה מחפש



פול א. הופמן

המדריך הטוב ביותר ל-Internet במהדורת כיס!

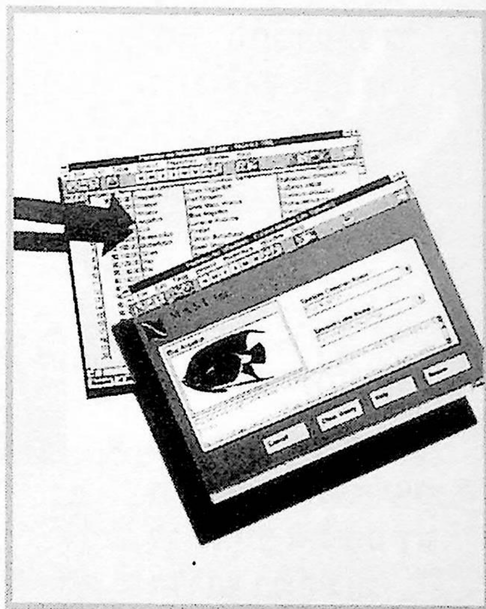




Paradox 4.5
למשתמשי Windows

מדריך בזק

מהדורה שנייה



- הספר המושלם למתחילים ולמנוסים כאחד
- מכסה את קבוצת העבודה Workgroup ושפת השאילתות Borland SQL Link
- ערוך לפי סדר האלף־בית האנגלי לנוחות המשתמש

לויי אנדרסון
קרי ג'נסן

מדריך הכיס המדויק והמקיף ביותר שקיים







מתאים
לגרסה העברית
והאנגלית

למשתמשים מתחילים
ולבעלי ניסיון

WINDOWS
כל אחד יכול
WINDOWS FOR DUMMIES

מאת אנדי ראת'בון

- דרך קלה ומהנה ללמוד אודות WINDOWS 3.1 ו-3.11
- ערכת "העזרה ראשונה" שלך להפעלה מוצלחת של WINDOWS
- מה לעשות כאשר מתעוררות בעיות - הסברים בעברית פשוטה

ספר העזר שנועד
לאנשים כמונו!

דף תמצית פקודות WINDOWS
כלול חינם בספר!

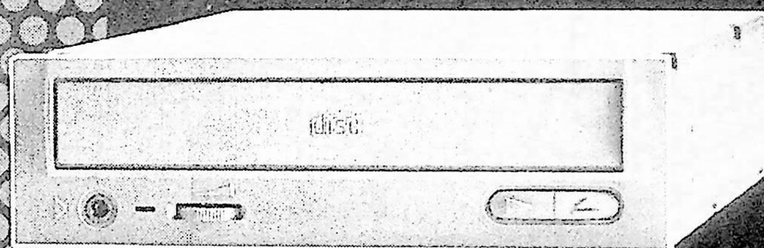


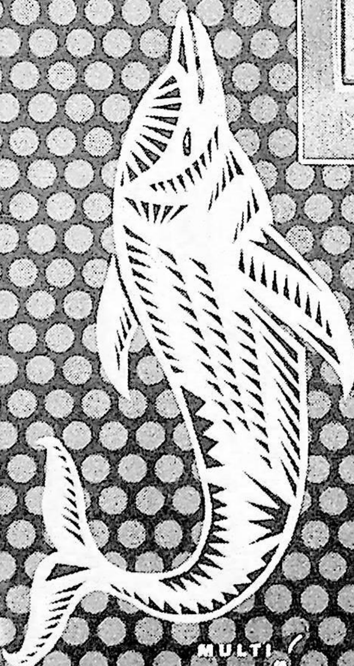
CD-ROM

8002 AT עם בקר

Dolphin 8000

סדרת כונני CD-ROM





- הכנה ל-CD-I MOVIES
- KARAOKE
- מהירות כפולה
- MPC 2-1 MPC 1
- 300K בייט / שנייה
- 270 ms זמן גישה
- MULTI-SESSION-1 SINGLE - PHOTO CD
- הכנה ל-CD-ROM XA
- ניהול צריכת חשמל
- כפתורי חזית של השמעה (PLAY) ודילוג (SKIP)
- איכות צליל גבוהה
- מגש ממונע



BUG MULTI SYSTEM

OPTICS STORAGE

הכל על
הדיסק הקשיח
מדריך להשרדות

מרק מינסי


כולל תקליטון
תוכנה חינם - תוכניות
שרות להצלת נתונים אבודים
סיוע צעד אחר-צעד
לבעיות ותקלות בכונן הקשיח
אמצעי מניעה נגד -
אובדן נתונים עקב הפסקות חשמל,
מחיקת נתונים באקראי
ואפילו וירוסים
למתחילים - רשימת עצות
לאיתור תקלות





C/C++ התוכנית הראשונה שלי

ספר לימוד C/C++ המוצלח ביותר למתכנתים מתחילים, שילוב של ספר ותקליטון שאינם פוסחים על דבר.

הספר כולל:

סקירה קלה להבנה, הממצה את עקרונות התכנות בשפת C - תוך זמן קצר ביותר תוכל לחבר תוכניות שימושיות

מאות דוגמאות, תרשימים ותוכניות הדגמה להבהרת החומר

מבחנים עצמיים ותרגילים בכל פרק, שנועדו לסייע לך לבסס את יכולת התכנות

עורך תוכניות וקומפיילר C מצורפים על גבי תקליטון – חוסכים לך את הצורך להשקיע כסף רב ברכישת תוכנה, ומאפשרים לך להתחיל מייד בעבודה

כל השיטות ותוכניות ההדגמה יפעלו בכל קומפיילר C/C++. הספר כולל סקירה בהירה של כל הנושאים המרכזיים בשפת התכנות C - החל מאופן ארגון התוכנית, דרך מבנים, מצביעים ועד לקובצי קלט/פלט – כולל הערות מיוחדות על מאפיינים ייחודיים של C/C++.

תקליטון 5 1/4 אינץ' המצורף חינם לספר, יסייע לך בלימוד התכנות על C/C++. התקליטון כולל קומפיילר C מלא, בצירוף הוראות שימוש; עורך תוכניות קל להפעלה; כל דוגמאות התכנות ותוכניות ההדגמה הכלולות בספר; קוד מלא של התוכניות; תשובות מלאות לכל התרגילים, ללימוד עצמי בקצב אישי.

ספר לימוד מעולה למתכנתי C/C++ למתחילים

✔ יסייע לך ללמוד את יסודות תכנות C/C++ - גם אם לא תכנתת מעולם

✔ שלל דוגמאות ותוכניות הדגמה לייעול הלימוד

✔ כל פרק מסתיים בשאלות למבחן עצמי, המסייעות לך לזכור את החומר הנלמד

✔ כולל תקליטון חינם המכיל קומפיילר C, עורך תוכניות ועוד

אודות המחבר

אלן ר. נייבאואר הוא בוגר ביה"ס וורטון באוניברסיטת פנסילבניה. הוא לימד תכנות במכללה, והיום הוא מחלק את זמנו בין כתיבה, ייעוץ ולימוד. הוא מחברם של ספרי הדרכה פופולאריים רבים בנושאי מחשבים ותכנות.





